SURVIVAL GUIDE FOR DENSE NETWORKS

BY

FARHANA ASHRAF

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2013

Urbana, Illinois

Doctoral Committee:

Associate Professor Robin H. Kravets, Chair and Director of Research
Professor Nitin H. Vaidya
Associate Professor Yih-Chun Hu
Professor Tarek Abdelzaher
Dr Ranveer Chandra, Microsoft Research

Abstract

High density affects the communication in an energy-constrained network since the energy-saving protocols are not designed to cope with the high contention expected in dense networks. While increased delay and reduced throughput are predictable results of contention from background traffic, high contention increases collisions, resulting in increased packet retransmissions and dropped packets. Two factors are mainly responsible for this high contention in the emerging networks. First, the networks are getting denser with the proliferation of smartphones and tablets. Second, the communication protocols used in these energy-constrained devices interact negatively with high density increasing contention even more. As a result, instead of saturating the bandwidth as expected with high traffic, the dense network starts thrashing resulting in a steep increase in delay and a drastic fall in throughput. The devices also end up consuming more energy since sleeping during the idle times is no longer an effective strategy to save energy. None of these effects is desirable.

The goal of our research is to improve communication in energy-constrained dense networks. While the increased density typically results in poor communication quality and high energy consumption, knowing that the network is dense often enables opportunities for optimization. Our research reveals that the increased number of devices and the frequent traffic, the characteristics that cause high contention in dense networks, can also be leveraged to improve the communication and energy-efficiency in these networks. With more devices within the communication range of a device, the neighbors can share the responsibility of finding an awake receiver in a wireless sensor network (WSN), or acquiring information about nearby access points (APs) to decide which AP to join in an infrastructure-based WLAN. The redundancy inherent in dense networks provides opportunities to reduce the delay in a multi-hop WSN by forwarding data packets to those neighbors which wake up the soonest. Interestingly, the security of the network can also be improved by leveraging the high connectivity. Neighbors can collaborate to eliminate any differentiability or predicatability in WSN communication such that the entire neighborhood appears to the jammer as a

single node with a big battery. Finally, local observation about the neighborhood can provide information about the contention in the network and enable the devices to adapt themselves to cope with high density.

The key to achieving efficient communication in an energy-constrained high density network is to embrace the high density. Our thesis shows that although high contention provides a harsh environment for communication, by leveraging the inherent characteristics of dense networks such as high connectivity, redundancy, frequent communication, devices can survive in such networks by successfully improving their quality of communication and energy-efficiency.

To My Family.

Acknowledgments

I am thankful to the Almighty for blessing me with a wonderful husband, a loving family and fabulous friends. This thesis would not have been possible without their inspiration and support. My husband Munawar Hafiz has been the continual source of support throughout the entire journey. Especially, his words of encouragement has kept me strong while we were apart for the past two years due to his job at Auburn University. He was never tired of making these seven hundred mile long frequent trips to Champaign. Thank you for being my friend who makes me happy and being my guide who can make any difficult situation much easier. I do not have enough words to thank my parents Dr. Mohammad Nurul Ashraf and Fatema Johora who are the constant sources of inspiration in every aspect of my life. Their dedication, love and advice have shaped my life. My sisters Ismehan and Nahian have always been there for me during the good times and the bad times. They have the amazing ability to brighten my day even when I am sad. I am thankful to all my friends for the amazing time we have spent together. I am also indebted to my colleagues at the Mobius group for their friendship, suggestions and support.

Finally, I thank my advisor Robin Kravets for her guidance and support. This PhD would not have been possible without her patient and diligent effort in shaping me as an independent researcher. I would like to extend my thanks to the committee members: Nitin Vaidya, Yih-Chun Hu, Tarek Abdelzaher and Ranveer Chandra for their guidance.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

With the proliferation of new wireless devices, the existing wireless networks are moving towards high density. Cisco has projected that there will be over 10 billion mobile-connected devices in 2017, exceeding the world's population at that time (7.6 billion) [5]. Naturally, with this many devices, there will be many areas with very dense networks. Wireless sensors are cheap; they are prone to failures. When sensors are used for critical applications such as forest fire detection or automated inventory management, many redundant devices are deployed in the networks to ensure reliability of communication. The density can reach upto 20 sensors per square meters. The same trend towards high density is also observed in WLAN environments. With the popularity of smartphones and tablets, the density we expect in public areas with open wireless access has changed dramatically. Examples of such high density networks include the wireless networks in large conferences, crowded auditoriums, sports stadiums, etc. In each of these environments, more devices induce more contention incurring increased delay and reduced throughput. The traditional energy-saving protocols further deteriorate the quality of communication since the high network density along with the strict energy constraints of the devices add new challenges to these networks.

The main problem with the traditional solutions for energy-constrained networks is that they were not designed considering the impact of possible high density. The communication protocols incur separate overhead for each device in terms of energy consumption and time. In a dense multi-hop wireless network such as WSN, this adds up to very high coordination overhead. First, the wakeup times of different devices may overlap. As a result, many devices contend during that short awake time window and the contention increases even more, voiding any benefit achieved from duty-cycling. Besides energy and delay issues, the devices also face severe jamming vulnerabilities when they follow the traditional duty-cycling solutions. The periodic wakeups induce a regular pattern of data transmissions, making the transmission times highly predictable. Jammers can take advantage of this opportunity and jam during the predicted transmission times while sleeping the rest of the time. As the network becomes denser, there is more traffic in the

network allowing the jammer to build its prediction model even faster. Hence, even an energy-constrained jammer poses a serious threat to the duty-cycled devices in a dense network.

In infrastructure-based dense WLANs, an interesting and even more devastating problem arises when the devices make any effort to improve the communication. The problem starts when devices decide to join a better network and so actively search each channel with probe packets to obtain access point (AP) information. Since the communication quality is always poor due to high levels of contention and frequent collisions, the mobile devices are constantly searching for better APs to associate with, overloading the networks to the extent where probing dominates and results in a network breakdown. Moreover, the synchronized channel accesses by these energy-constrained devices add more contention to the already overloaded network, resulting in spikes of contention at the beginning of each wakeup. The common result is that devices are not able to send much data and, at the same time, their batteries drain at an increased rate that is not proportional to their successful data.

The goal of our research is to tackle these challenges and make dense networks work for the user. While increased density typically is at the root of poor communication quality, knowing that the network is dense often enables opportunities for optimization. In our research, we explore which inherent characteristics of high density networks can be leveraged to improve communication and energy-efficiency in these networks. For example, the high connectivity of devices can be used to opportunistically improve communication. As the network becomes denser, each device has more neighbors within its communication range. The frequent traffic from the neighbors exposes information about their wakeup schedules providing opportunities for low cost coordination in WSN. Moreover, the inherent redundancy in the dense network often results in multiple paths with the same cost available between the same source-destination pair offering opportunities for reducing delay. The devices can even collaborate to collectively defend against a jammer by obscuring their data transmission patterns within the neighborhood. The opportunities for improvements are not limited to WSN. The dense WLANs can also take advantage of the high connectivity. In dense networks, the nearby devices have similar AP fingerprints which allow them to obtain AP information opportunistically. Finally, by distributed managing of each power-saving device based on local observation about the neighborhood, contention can be controlled preventing the imminent network breakdown. Hence, we can summarize that the key to ensuring survival in dense networks is to embrace density.

In our dissertation, we propose that: *communication in an energy-constrained high density network can*

be improved by leveraging the inherent characteristics of the dense network. The increased connectivity and frequent traffic can be leveraged to achieve low cost coordination between senders and receivers in a dense WSN and to provide low overhead AP discovery in a dense WLAN. The neighbors can collaboratively strategize to defend against a reactive jammer, whereas the redundancy of paths can be exploited to reduce delay in a dense WSN. Finally, by locally managing each energy-constrained device, contention can be mitigated in a dense WLAN.

1.1 Research Contributions

The thesis can be divided into five parts. In each part, we focus on a specific challenge of dense networks, e.g., energy-efficiency, latency, jamming resilience, AP discovery and high contention, and solve it by leveraging characteristics of the dense neighborhood. The target for the first three approaches are densely deployed WSN. Although the problems are discussed in the context of WSN, such problems are generic to any energy-constrained dense wireless network without any infrastructure support. Hence, the proposed solutions can be applied to any such networks. The last two approaches target improving communication in dense infrastructure-based WLANs.

1.1.1 Energy-efficient Communication

Energy-constrained networks typically conserve energy by following a periodic wakeup-sleep schedule. To coordinate and exchange data, the current energy-saving MAC protocols for WSN either require tight synchronization between the neighbor wakeup schedules or expend a significant amount of energy in signaling the sleeping nodes. High density makes it difficult to synchronize the wakeup schedules of all nodes in the neighborhood and also incurs high overhead when many neighbors have to send long signals to wakeup their receivers. The main problem is that each device acts as an individual and the protocols ignore the fact that there are often multiple sender-receiver pairs within the same neighborhood in a dense network.

In response, we designed Neighborhood Power Management (NPM), a low cost communication protocol for duty-cycled networks that amortizes the cost of coordination among multiple nodes in the neighborhood. NPM enables all awakened sender-receiver pairs to communicate without requiring separate wakeup signals for each pair. Additionally, NPM uses wakeup information obtained from its data-driven synchronization mechanism to dynamically adapt the signal lengths. In denser networks, NPM has more

opportunities to reduce its per-node coordination overhead, further improving the energy-efficiency of the communication.

1.1.2 Bankrupting the Jammer

Traditional MAC protocols for WSN were designed to achieve energy-efficiency, without considering the presence of a jammer. As a result, each packet has a low jamming resilience, the types of the packets are easily differentiable, and the data transmission times are highly predictable. Given enough traffic, an intelligent jammer can take advantage of such information to formulate energy-efficient jamming attacks. In dense networks, the frequent communication facilitates the jammer to build its prediction model faster.

We designed Jam-Buster that improves the jam-resilience of the system by obscuring the otherwise exposed wakeup schedule information and traffic patterns. Without such information, the entire neighborhood appears to be a single entity to the jammer, preventing it from achieving energy-efficient selective jamming. As the network becomes denser, the jammer has to transmit more, reducing its own network lifetime and enabling easy localization of the jammer.

1.1.3 Reducing Delay with Anycast

In duty-cycled networks, sleep latency is one of the major sources of delay. This is the time a sender must wait for its receiver to be awake before transmitting the data packets. In densely deployed networks such as RFID networks, there are often multiple shortest hop paths between the same source-destination pair. When different nodes in the network wake up at different times, these multiple paths provide a great opportunity to reduce the sleep latency at each hop. The traditional protocols do not utilize this opportunity.

We designed Any-MAC, a MAC layer forwarding protocol that exploits this inherent redundancy to improve delay in asynchronous duty-cycled networks. At each hop, Any-MAC forwards data to the neighbor that wakes up the soonest and hence opportunistically improves its delay performance. Any-MAC is a generic plug-in that can be applied to any duty-cycled MAC protocol to enable anycast and achieve lower delay.

1.1.4 Opportunistic AP Discovery

Mobile devices overload an infrastructure-based dense WLAN with probe packets in their effort to improve communication by switching to a better AP. Unfortunately, simply finding a new AP does not solve the problem. Since all nearby APs are probably overloaded and so do not provide any better channel conditions, devices experience no improvement even after successful handoffs. Probing does not stop and the flood of AP probes continue to increase contention in the network. We discovered this devastating impact of high density on the process of AP discovery by analyzing the traces collected at `INFOCOM 2012`.

In dense networks, the surrounding APs of the nearby devices are often the same. However, the traditional AP discovery protocols do not exploit this similarity. We designed Dynamo-Probing to exploit these similar AP fingerprints and the frequent scans in dense networks to opportunistically obtain AP information by listening to the results of nearby active scans. By eliminating the unnecessary probe storms, Dynamo-Probing reduces contention in the networks and results in improved quality of communication.

1.1.5 Managing Contention with PSM

The wakeup times of the energy-constrained devices in a WLAN are synchronized with the beacons sent from their APs. Hence, all devices with data buffered at the AP contend for channel accesses right after receiving a beacon. In dense networks, these synchronized channel accesses add more contention to the network, resulting in contention spikes at the beginning of each beacon interval. The biggest problem with the existing protocols including the IEEE 802.11 Power Save Mode (PSM) is that they are agnostic of network conditions, hence they not only fail to improve energy-efficiency, but result in increased contention.

We designed Skip-stop, a PSM-based energy management approach that adapts to contention in dense networks by skipping AP polling during some beacon intervals and so limiting the number of devices active within each beacon interval. The ultimate goal of Skip-stop is to avoid introducing additional contention and to rein in the energy consumption to match the device's data transmissions.

1.2 Thesis Structure

The remainder of this dissertation is organized as follows. Chapter 2 presents our solution to energy-efficient communication in dense networks utilizing the high number of neighbors and the exposed schedule informa-

tion to optimize coordination overhead. Chapter 3 describes the collaborative defense against an intelligent jammer to force the jammer to spend more energy and be easily detected. Next, in Chapter 4, we present our approach of exploiting redundant paths in dense networks to reduce delay. In Chapter 5, we describe an opportunistic method of obtaining AP information in dense networks by exploiting nearby active scans. We propose an energy-management approach for WLAN in Chapter 6 to manage PSM and control contention in the networks. Finally, we present our research plan and conclude in Chapter 7.

Chapter 2

Neighborhood-based Power Management

Coordination between a sender and its receiver is essential to enable successful data transmission in any duty-cycled network. Achieving energy-efficient coordination in dense wireless ad hoc and sensor networks is particularly challenging since the increased number of neighbors not only makes it difficult to synchronize the wakeup schedules of all nodes in the neighborhood, but also incurs high overhead when many neighbors have to send long signals to wake up their receivers. However, the frequent traffic in dense networks also provides the nodes opportunities to optimize their coordination overhead. A simple data-driven synchronization can enable the nodes to achieve much of the benefits of full knowledge of wakeup schedules. Moreover, the wakeup signals often wake up many neighbors. This provides the awakened nodes opportunities to send their data packets without requiring additional wakeup signals and thus sharing the overall coordination cost among neighbors. Our main contribution is the design of *Neighborhood-based Power Management* (NPM), an energy-efficient MAC protocol that integrates opportunistic sending and opportunistic synchronization, to achieve energy-efficient coordination in dense networks.

2.1 Challenges for Energy-efficient Communication

The main challenge of designing an energy-efficient communication protocol for a duty-cycled network is to coordinate the senders and the receivers efficiently. Given the dynamic communication load expected in wireless sensor networks (WSNs), the coordination must be energy-efficient for both high and low traffic. The most common target for energy conservation is the wasted idle time during communication. In an ideal network, senders and receivers would wake up at the same time, transmit their data, then go back to sleep. In real networks with clock drift and limited energy reserves, nodes can compensate by using synchronization to ensure that both the sender and the receiver know when to wake up and signaling to know when to communicate.

Synchronization-based coordination requires the exchange of schedule information between neighbors so senders can transmit exactly when the receiver is going to be awake. However, clock skew quickly results in a loss of synchronization, resulting in the need to periodically exchange schedule information [89, 82, 91, 74]. DW-MAC assumes that an out-of-band synchronization protocol will be used to synchronize the clocks of the neighboring nodes during its synchronization period. Synchronization-based protocols can either use dedicated synchronization messages or piggyback synchronization information onto data messages. For example, IEEE 802.11 PSM [24] uses a dedicated beacon for synchronization. However, since the frequency of PSM's beacon message exchange is not tied to the traffic generation rate or the clock skew, nodes send beacons even when there is no data to send, causing unnecessarily high synchronization costs at low rates and small clock drift. In response, S-MAC [89] and T-MAC [82] are able to choose the length of their synchronization period based on clock drift. Additionally, synchronization information can be piggybacked onto data messages whenever possible. While this works well at high traffic rates, when the traffic rate is not high enough to handle the clock drift, nodes must still send dedicated synchronization messages, resulting in high synchronization overhead.

Signaling-based coordination can be used in networks where nodes are completely unaware of the wakeup schedules of their neighbors. The sender's signal lets the receiver know that there is a transmission ready. The most straightforward approach is to send signals (i.e., preambles) at least as long as a channel polling interval (e.g., B-MAC), enabling receivers to periodically wake up and poll the channel. When receivers detect radio activity, they remain awake waiting for data transmissions. In general, signaling costs can be controlled by varying the channel polling interval. However, while more frequent channel polling can reduce signal length, it results in increased listening overhead for the receivers. B-MAC provides optimized channel polling intervals using knowledge of network size and traffic patterns. However, without a priori knowledge or in networks with dynamic traffic patterns, it is difficult to find optimal parameters, potentially resulting in ineffective energy management, reduced network throughput and increased packet delay. Alternatively, sender nodes in X-MAC [20] and SpeckMAC [83] send short strobed signals that include the address of the intended receiver. A receiver acknowledges the signal to stop the signal transmission and to initiate the actual data transmission, shortening wakeup signals, while still using effective polling periods. However, this reduction is achieved at the cost of high idle listening costs at low traffic rates. Since a sender switches between transmit and listen mode during the entire preambling period, a receiver needs to poll the

channel longer to detect the signal. X-MAC further reduces the signaling overhead by allowing any node to send all queued packets to the same receiver using a single signal. However, to apply this optimization, the traffic must be bursty. For periodic traffic, nodes using X-MAC still transmit one wakeup signal per data message. Thus, despite shortened preambles and support for traffic bursts, the signaling overhead is still very high at all rates for periodic traffic.

Hybrid protocols, like SCP [91] and AS-MAC [44], combine both synchronization and signaling mechanisms to coordinate the senders and the receivers. By assuming loose synchronization between the wakeup schedules of the neighboring nodes, these hybrid protocols can use a larger synchronization period to reduce their synchronization overhead. Although the loose synchronization necessitates the use of signaling, the knowledge obtained from the synchronization complements the signaling mechanism by allowing the protocols to have shorter signals (i.e., preambles). Even with the loose synchronization requirement, both SCP and AS-MAC require a short synchronization period in networks with any significant clock drift. Additionally, at low traffic rates, these protocols incur very high synchronization overhead, since there is little opportunity for piggy-backing synchronization messages. Moreover, SCP uses the synchronization information to synchronize the wakeup schedules of its neighbors, which results in more contention in the network and increased delay. AS-MAC, on the other hand, maintains an asynchronous wakeup schedule and uses the synchronization information to directly reduce the length of the preambles. However for AS-MAC, an additional synchronization overhead is incurred since each node now performs periodic neighbor discovery to allow new nodes to join the network and obtain wakeup schedule information about neighbors. Besides the synchronization overhead, the signaling costs for SCP and AS-MAC are also significant at high data rates, since signaling is done on a per data message basis.

In response to these limitations, we present Neighborhood-based Power Management (NPM). The design of NPM is based on two observations. First, high synchronization overhead stems from the periodic exchange of schedule information, even when there is no data to transmit. While knowledge of wakeup schedules can greatly reduce energy consumption, such knowledge can be obtained during active communication through piggy-backed timing information. Use of such *opportunistic synchronization* in dense networks can achieve much of the benefits of full knowledge of wakeup schedules. Second, the main reason for high signaling costs is the inefficient use of the signal. Although a signal wakes up all nodes that can hear it, only the sender-receiver node pair are allowed to exchange data. All other nodes go back to sleep,

even if they have data to transmit. By allowing the signal to wake up and enable *opportunistic sending* for all nodes that hear the signal, signaling costs can be amortized over multiple transmissions from multiple nodes, while incurring only minor additional synchronization costs. Hence, NPM is designs to balance the synchronization and the signaling overhead by leveraging the increased number of neighbors and the frequent communication in dense networks.

2.2 Neighborhood-based Power Management (NPM)

Neighborhood-based Power Management (NPM) is a hybrid protocol that utilizes both signaling and synchronization mechanisms to coordinate nodes before a data exchange. Since nodes in NPM exchange schedule information only when there is data, the synchronization cost is low. Such synchronization can drift not only at high clock drift but also at low traffic. To account for such loose synchronization, nodes send wakeup signals (i.e., preambles) before sending their data messages to make sure that the receivers are awake. However, unlike other signal-based protocols, NPM's preamble enables all nodes in the neighborhood of the signaling node to coordinate and exchange data, amortizing the signaling cost over multiple data transmissions. Nodes further reduce signaling costs by dynamically adapting preamble length using schedule information obtained from the low-overhead loose synchronization mechanism. In this section, we describe the two signaling mechanisms adopted by NPM: neighborhood wakeup and adaptive preambling. We also describe the details of NPM's beacon-based data-driven synchronization.

2.2.1 Signaling Mechanisms

All nodes in NPM wake up periodically and poll the channel for activity to receive incoming data messages. Due to the imperfect (out-of-date) synchronization information available to the nodes, senders in NPM need to signal the receivers to remain awake until they receive their data messages. The control overhead due to signaling can be high in a network with unsynchronized wakeup schedules. NPM integrates two mechanisms to reduce this overhead. While neighborhood wakeup reduces the total number of preambles required to send the data messages by allowing multiple data transmissions per preamble, adaptive preambling reduces the length of each preamble by utilizing the available schedule information about the neighborhood.

Neighborhood Wakeup

When nodes in NPM detect a preamble during their regular periodic wakeup, they remain awake until the end of the preamble to receive any data messages directed towards themselves. The key idea behind neighborhood wakeup in NPM is that all awakened nodes should be able to stay awake and transmit data, if they have any. In this way, a single preamble could serve for multiple communications between nodes in the same neighborhood. By using one preamble for multiple data transmissions, NPM can achieve lower signaling overhead as follows:

- If a node has a high amount of traffic to a single receiver, the node only needs one preamble to transmit multiple messages to that receiver;
- If a node has multiple messages for different receivers in its neighborhood, the node only needs one preamble to wake up all receivers and transmit its messages;
- If the preamble wakes up both the sender and the receiver of another flow, these nodes can exchange their data without initiating a new preamble.

NPM divides communication time into three components, the signal (i.e., preamble), the control window and the data window, which together form a *transmission phase* (see Figure 2.1). After detecting a preamble, all nodes in the neighborhood of the preambling node coordinate to complete their data transmissions. Since the receiver for any of the awakened neighbors may be out of transmission range of the preambling node, some receivers may not have been awakened by the preamble. Therefore, senders exchange *signal messages* with their intended receivers during the control window (similar to the ATIM window of IEEE 802.11 PSM). Nodes that have no data to send or receive, or cannot reach any nodes they have packets for, quickly go back to sleep after the control window.

After the control window, all source-destination pairs complete their data transmissions using CSMA/CA in the data window. To support short term bursts of traffic, the nodes remain awake through the entire data window enabling NPM to opportunistically send newly generated data messages within the same data window without additional signaling. Transmissions that fail during opportunistic sending are rescheduled in the next transmission phase.

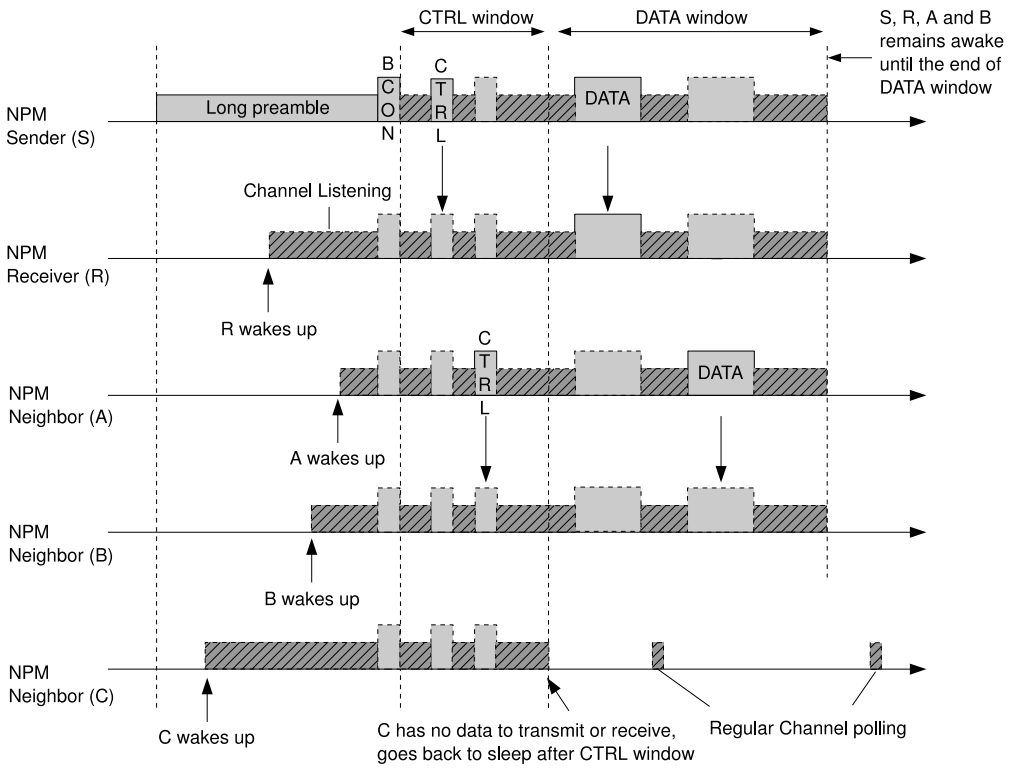

Figure 2.1: Neighborhood wakeup with full preamble

Adaptive Preambling

Nodes in NPM adopting the basic neighborhood wakeup mechanism always send full length preambles (we refer to this protocol as *NPM full*) (see Figure 2.1,2.2). The *adaptive preambling* component of NPM utilizes the available loose synchronization information to dynamically adapt the length of the preambles. Nodes only reduce the length of their preambles when they have recent synchronization information about their receivers. However, when the available synchronization information is out-of-date, nodes must transmit full length preambles. The adaptive preamble mechanism is also never applied to broadcast messages since there is no guarantee to wake up all nodes in the neighborhood if the preamble length is shorter than the nodes' sampling interval. To account for clock drift, nodes add a guard time to both ends of their preambles.

NPM supports three different approaches for adapting the length of the preambles:

- *NPM targeted* aims to wake up only the receiver of the signaling node. Since nodes transmit their preambles during the expected wakeup times of their receivers, *NPM targeted* achieves the shortest preambles (see Figure 2.3).

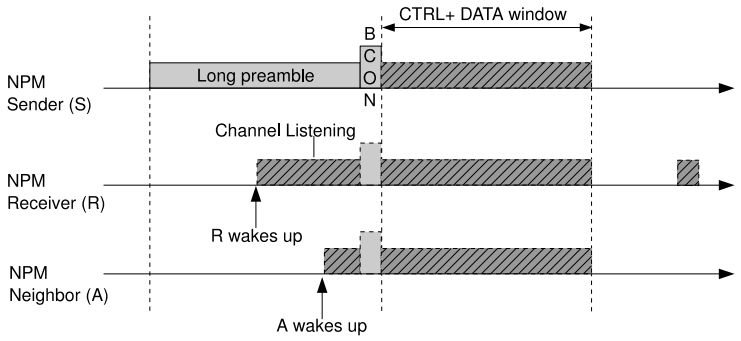

Figure 2.2: NPM-Full Approach

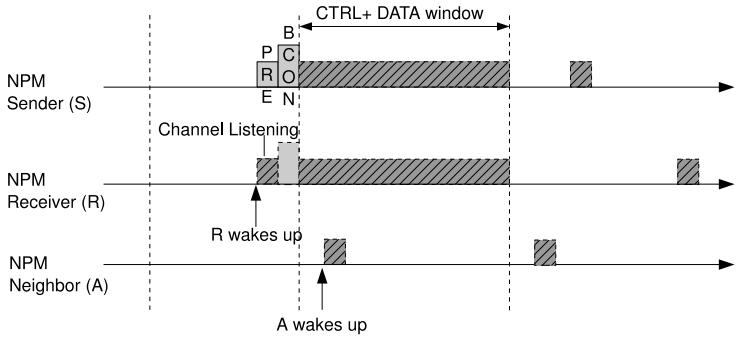

Figure 2.3: NPM-Targeted Approach

- *NPM finish early* aims to improve delay performance while utilizing the neighborhood wakeup mechanism. Thus, nodes stop transmitting preambles as soon as their receivers wake up, similar to X-MAC (see Figure 2.4).

- *NPM coverage* aims to fully utilize the neighborhood wakeup mechanism, while still reducing the length of the preambles. Preambles are long enough to wake up all known nodes in the neighborhood (see Figure 2.5).

By reducing the length of the preambles, adaptive preambling decreases the potential use of the neighborhood wakeup and opportunistic sending components of NPM. Essentially, the shortened preambles in *NPM targeted* wake up fewer nodes, increasing the total number of preambles required to complete all data transmissions. On the other hand, *NPM coverage*, which benefits the most from the neighborhood wakeup, has the longest preambles. The choice of the approach that most suits the user's needs is the results of a tradeoff between energy efficiency and latency, and will be thoroughly explored in Section 2.4.

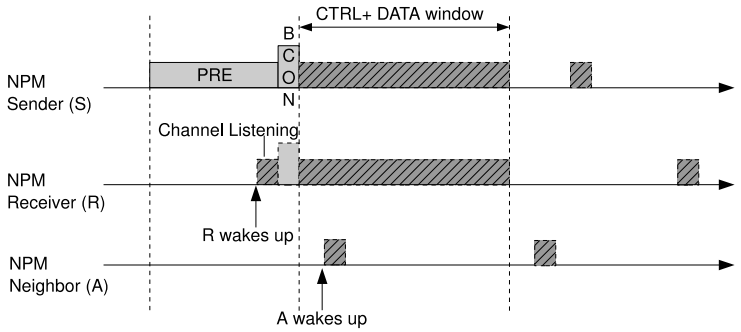

Figure 2.4: NPM-Finish Early Approach

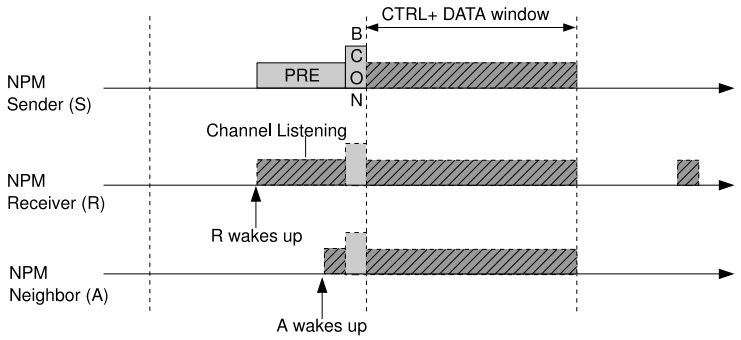

Figure 2.5: NPM-Coverage Approach

2.2.2 Synchronization Mechanisms

NPM uses beacon-based synchronization to obtain the wakeup schedule information. Instead of exchanging schedule information periodically (like S-MAC, SCP-MAC and IEEE 802.11 PSM) and spending a fixed overhead for synchronization, nodes in NPM exchange their schedule information only when there is data to send. This data-driven synchronization approach saves NPM from wasting energy on synchronization when there is no traffic in the network.

Schedule information (i.e., synchronization) is exchanged via two types of control messages: beacons and signal messages. Instead of sending a beacon at the beginning of each wakeup period and thus incurring a fixed synchronization overhead like IEEE 802.11 PSM, NPM nodes send one beacon message per transmission phase. The preambling node broadcasts the beacon message just after sending the preamble (see Figure 2.1) and thus provides its schedule information to all awakened neighbors. Due to neighborhood wakeup, the synchronization overhead due to beaconing is amortized over multiple data transmissions. Nodes in NPM also piggyback their schedule information when they reply to the signal messages during the

Algorithm 1: Determining the preambling node for NPM

```
if queue_len ≥ QUEUE_THRESHOLD then
    backoff = random(1, QUEUE_LEN_SLOT)
else
    if age ≥ AGE_THRESHOLD then
        backoff = random(1, AGE_SLOT)
    else
        if preambles ≥ PREAMBLES_THRESHOLD then
            backoff = random(1, PREAMBLES_SLOT)
        else
            backoff = random(1, DEFAULT_SLOT)
        end
    end
end
```

control window. Although the synchronization overhead due to signal acks increases as more data messages are transmitted, the size of the overhead is very small (8 bytes for the schedule information).

Synchronization information is sent as the time difference between the local time of the sender and the time when the sender will wakeup for the next cycle according to its periodic wakeup schedule. Thus, the schedule information is not affected by the clock skews of the different nodes in the neighborhood. Each node keeps track of the synchronization information of each of its neighbors by maintaining a *schedule table*. Each entry in the schedule table contains two pieces of information: the time difference between the neighbor's schedule and the local node's own wakeup schedule, and the age of the entry. Since synchronization information is not exchanged periodically, this information can become stale and the age information is used to purge stale data.

2.2.3 Fairness and Load Balancing Issues

When multiple nodes in the neighborhood have data to send, any one of them can act as the *preambling* node since a preamble wakes up all nodes within transmission range. To avoid preamble collisions, one of the nodes is chosen randomly as the *preambling node*. However, a random choice may cause some flows to starve or may drain more energy from some of the nodes in the network. NPM adopts a weighted random backoff mechanism to enforce fairness and load balancing. All potential senders back off for a random time period, and the node with the lowest backoff wins the contention and becomes the *preambling* node for the entire neighborhood. NPM uses the following parameters to choose the random backoff period:

- Number of packets in the queue;
- Age of the oldest packet in the queue;
- Number of preambles sent.

NPM achieves load balancing by distributing the responsibility of preambling among all potential senders within the neighborhood. By choosing the node having more packets in the queue as the *preambling* node NPM can guarantee higher throughput. Since biasing the choice of preambling node based solely on queue length can lead to starvation for nodes with lower traffic generation rate, NPM also considers the age of the oldest packet in the queue. Finally, NPM balances the load across nodes by considering how frequently each node has been the preambling node. Algorithm 1 describes the algorithm for choosing the backoff time for an NPM preamble.

2.3 Analytic Model with Periodic Traffic

In this section, we describe the energy model of NPM and four other protocols: B-MAC, X-MAC, IEEE 802.11 PSM and SCP. Our analysis shows how parameters such as network density, data generation rate and duty cycle influence the performance of each protocol. We model a single hop network with $N + 1$ nodes, which provides insights into the expected protocol behavior in larger networks. We model CBR traffic, where each of the $N + 1$ nodes generate r packets per second. We analyze energy efficiency by comparing the energy consumed by each protocol for sending M data messages.

Total energy, E_{tot}, depends on the time that the radio transceiver spends in each of the four different power states, *transmitting*, *receiving*, *idling* and *sleeping*. We denote the power in both *transmitting* and *receiving* state as P_a. and the idle power as P_i, with $P_a >> P_i$. We ignore the (very small) power required to keep the transceiver in *sleep*. Thus, E_{tot} is a combination of active energy, $E_a = P_a \cdot t_a$, and idle energy, $E_i = P_i \cdot t_i$, where t_i and t_a are the time spent in idle and active states. To minimize E_{tot}, a protocol has to minimize both E_a and E_i. Idle energy E_i can be minimized with low duty-cycles. However, minimizing active energy, E_a, requires minimizing its two components: data transmission energy, E_d, and control overhead, E_c. Since E_d must be incurred to transmit data, we focus on minimizing E_{tot} by minimizing E_c.

Overall, E_c has three major components: E_{pre} for preambling, E_{sync} for synchronization and E_{sig} for

exchanging control signals before sending the actual data messages.

$$E_c = E_{sync} + E_{pre} + E_{sig}$$

The overheads are determined by: the rate at which each protocol exchanges control messages and the number of awake nodes during the exchange.

Preambling overhead, E_{pre}, which is used by B-MAC, X-MAC, SCP and NPM, depends on the length of each preamble, t_{pre}, and the number of preambles, n_{pre}, used for sending M data messages. Since any node that wakes up during a preamble continues to receive the rest of the preamble, preamble length, t_{pre}, determines the number of awake nodes, A. The portion k of the entire preamble that the receiver actually receives is, however, determined by the synchronization between the wakeup schedules.

$$E_{pre} = n_{pre} \cdot (1 + k \cdot A) \cdot t_{pre} \cdot P_a$$

B-MAC wakes up all N nodes in the network by using a T_{wup} long preamble, where T_{wup} is the wakeup period of the nodes in the network. However, SCP can wake up all nodes using short preambles due to its loosely synchronized network. SCP's synchronization also causes each node to receive the entire preamble ($k = 1$). Whereas, in an unsynchronized network (such as for B-MAC and NPM), each awake node receives on average only half of the preamble ($k = 0.5$). The value of A in NPM depends on the node's duty cycle and the length of the preamble t_{pre} used as will be explained later in this section. For periodic traffic, B-MAC and SCP uses one preamble for each of its M data messages. NPM however amortizes the E_{pre} over multiple transmissions by sending one preamble per transmission phase. Assuming NPM sends R_d data messages per transmission phase, $n_{pre} = \frac{M}{R_d}$ preambles are needed to transmit M messages.

$$\begin{aligned} E_{pre}^{scp} &= M \cdot (1 + N) \cdot t_{pre} \cdot P_a \\ E_{pre}^{bmac} &= M \cdot (1 + \frac{N}{2}) \cdot T_{wup} \cdot P_a \\ E_{pre}^{npm} &= \frac{M}{R_d} \cdot (1 + \frac{A}{2}) \cdot t_{pre}^{npm} \cdot P_a \end{aligned}$$

The strobed preambles in X-MAC result in an average preamble length of half of the maximum preambling time, $T_{wup}/2$. Half of this time is spent in sending preambles, while the other half is spent listening for an acknowledgment.

$$E_{pre}^{xmac} = M \cdot \left(\frac{T_{wup}}{4} \cdot (P_a + P_i) + (\frac{N}{2} + 2) \cdot t_{pre} \cdot P_a \right)$$

For all protocols, E_{pre} increases as the network density N increases. NPM consumes the least E_{pre} since it amortizes the cost of preambling over multiple data transmissions. BMAC uses the highest energy due to the long preamble used. The comparison of preambling energy between SCP and NPM will depend on the length of the preamble used and the value of A. For NPM *full*, $A = N$ and $t_{pre} = T_{wup}$, increasing energy consumption. However, the value of R_d increases which in turn reduces E_{pre}. If the rate r is relatively high that $R_d > 1$, NPM *targeted* uses much less energy than SCP.

To determine the value of A, the number of nodes awakened by a preamble of length t_{pre}, assume a uniform distribution for the wakeup schedules across a wakeup period T_{wup}. If each node remains awake for T_l and asleep for T_s within each duty cycling period, the probability p that a preamble of length t_{pre} wakes up a particular node is defined as follows: $p = \min\left(\frac{T_l+t_{pre}}{T_l+T_s}, 1\right)$. Thus the average number of nodes awakened by such a preamble is defined as: $A = 1 + p \cdot (N-1)$. For NPM *full*, the T_s long preamble wakes up all nodes in its neighborhood. The remaining NPM protocols wake up fewer nodes using their shorter preambles, and thus reduce the energy spent due to preambling.

The next major component of E_c is synchronization, which is used by IEEE 802.11 PSM, SCP and NPM. Nodes synchronize by exchanging beacon messages of length t_b. Assuming that n_b beacon messages are required and A_b nodes are awake during the transmission of a beacon, the energy overhead for synchronization E_{sync} is:

$$E_{sync} = n_b \cdot (A_b + 1) \cdot t_b \cdot P_a.$$

In SCP and NPM, the number of nodes that receive the preamble and the beacon are the same, thus $A_b = A$. In PSM, all N nodes wake up at the beginning of the control window and receive the beacon. PSM sends one beacon per wakeup period, thus n_b does not depend on the data generation rate r, and it wastes energy even in the absence of data traffic. However, the high clock drift in wireless sensor networks causes the beaconing of PSM to be insufficient. Depending on the traffic rate r, SCP piggybacks synchronization information on each of its data messages, causing minimal overhead. However, when the traffic rate r is low, SCP needs to send separate synchronization messages. In that case, n_b for SCP depends on the synchronization period T_{sync}. NPM sends one beacon per transmission phase, requiring a total of $\frac{M}{R_d}$

beacons.

$$E_{sync}^{psm} = n_b^{psm} \cdot (N+1) \cdot t_b \cdot P_a$$
$$E_{sync}^{scp} = n_b^{scp} \cdot (N+1) \cdot (t_{pre} + t_b) \cdot P_a$$
$$E_{sync}^{npm} = \frac{M}{R_d} \cdot (A+1) \cdot t_b \cdot P_a$$

Thus, we observe that unlike other protocols, NPM's synchronization overhead is traffic-driven and is amortized over multiple data transmission similar to the preamble energy.

E_{sig} is used in protocols (such as IEEE 802.11PSM and NPM) that allow transmissions from multiple nodes within a single transmission phase. Nodes coordinate by exchanging control signals of length t_{sig} during a control window, T_a. According to our traffic model, each of the $A+1$ awake nodes send control signals to their receivers. However, only the flows that have both of their sender-receiver pair awake receive signal acknowledgements back from their receivers. The number of such flows, F, depends on the previously defined probability p: $F = 1 + p + p^2 \cdot (N-1) = 1 + A \cdot p$.

After transmitting the control signals, nodes stay idle during the remaining control window T_a. The total idle energy consumed during the control signaling depends on the total number of transmission phases used to send M messages. The number of transmission phases is determined by the total number of messages R_d sent during one transmission phase.

$$E_c^{npm} = \frac{M}{R_d} \left((1+A) \left((1+A) t_{sig} + F t_{sig}^{ack} \right) (P_a - P_i) + T_a P_i \right)$$
$$E_c^{psm} = M \cdot (1+N) \cdot (t_{sig} + t_{sig}^{ack}) \cdot (P_a - P_i) + \frac{M}{R_d} \cdot T_a \cdot P_i$$

R_d, the total number of data messages transmitted per data window, depends on F, the number of nodes that remain awake during the data window of length T_d. Each node transmits all data messages that they have accumulated since their last active data window. With a total of $N+1$ nodes in the network and F active nodes within each data window, each node has to wait $((N+1)/F) - 1$ transmission phases before its active data window arrives. Thus, R_d is the total number of data messages accumulated over a time period of $(N+1)/F$ transmission windows by each of the F nodes.

$$R_d^{npm} = \mathtt{max}(r \cdot (N+1) \cdot (t_{pre} + t_b + T_a + T_d), 1)$$
$$R_d^{psm} = r \cdot (N+1) \cdot (T_a + T_d)$$

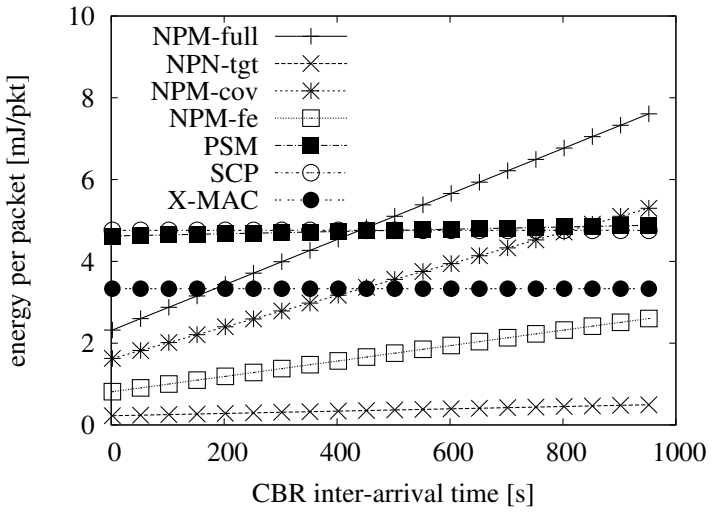

Figure 2.6: Energy per packet described by analytic model for single-hop network

NPM avoids the unnecessary idling during the control window when the data generation rate r is low. Also, NPM reduces the control overhead by sending fewer control signals. Since fewer nodes are awake in NPM, the receiving cost is also less.

E_d for IEEE 802.11 PSM and NPM is calculated similar to E_c. NPM consumes similar energy as PSM when exchanging data messages in the data window T_d. The higher value of R_d relative to PSM, allows NPM to use fewer transmission windows to transmit M messages, and thus consume less energy idle listening during its data window.

$$E_d^{psm} = M \cdot (N+1) \cdot (t_d + t_{ack}) \cdot (P_a - P_i) + \frac{M}{R_d} \cdot T_d \cdot P_i$$
$$E_d^{npm} = M \cdot F \cdot (t_d + t_{ack}) \cdot (P_a - P_i) + \frac{M}{R_d} \cdot T_d \cdot P_i$$

For protocols that do not allow multiple transmissions from different nodes within a transmission window (such as X-MAC, SCP, B-MAC), E_d depends on F, the number of nodes awake during data transmission.

$$E_d^{xmac} = M \cdot 2 \cdot (t_{data} + t_{ack}) \cdot P_a$$
$$E_d^{bmac,scp} = M \cdot (N+1) \cdot (t_{data} + t_{ack}) \cdot P_a$$

To demonstrate the expected energy consumption as derived in our model, we plotted the total energy per packet (i.e., the sum of each contribution) spent by each of the protocols in a single-hop network of 10

nodes in Figure 2.6. B-MAC is omitted from this graph to make it more readable, since B-MAC's values are several times higher than the competitors. As expected, the energy consumption for SCP and X-MAC is constant since both protocols incur a fixed overhead per message. However, SCP requires more energy from overhearing the synchronized wakeup. In comparison, PSM and all of the NPM approaches decrease linearly with increasing message rate. Although NPM is more efficient with high traffic rates, the NPM *full* and coverage policies suffer from excessive preambling in low traffic conditions. Although PSM looks very promising even from this model, it is important to note how this model does not account for clock drift that would cause PSM to require additional synchronization overhead. At very low data rates, it is clear from this graph that finish early and targeted are more expensive. However, we will show in the next section that in realistic networks, the crossover points are at very low data rates and that for bursty traffic, the crossover points disappear. In the next section, we present our simulation results to verify the expected energy consumption shown in our model and to evaluate the energy efficiency and performance of all of the protocols in larger multi-hop networks.

2.4 Evaluation

The goal of our evaluation is to show that by maintaining loose data-driven synchronization, NPM can achieve energy-efficient communication without affecting performance (i.e., delay, throughput). To demonstrate these benefits, we analyze the contributions of the two signaling mechanisms of NPM, neighborhood wakeup and adaptive preambling, in reducing the energy waste due to coordination. As a baseline for our evaluation, we compare NPM to signaling-based protocols: B-MAC and X-MAC, hybrid protocol: SCP, and synchronization-based protocol: the IEEE 802.11 PSM. It is important to note that IEEE 802.11 PSM is designed for an ideal network with perfectly synchronized wakeup schedules and we compare NPM to PSM only to understand how NPM in a real unsynchronized network performs compared to PSM working in a perfectly synchronized network. To verify the effectiveness of the neighborhood wakeup, we compare the NPM *full* approach to its dual B-MAC and the NPM *finish early* approach to its dual X-MAC. The comparison between NPM *finish early* and X-MAC is particularly interesting, since the burst support in X-MAC is similar to the neighborhood wakeup of NPM, but has a more constrained criterion compared to NPM. The combined effect of neighborhood wakeup and adaptive preambling is revealed when we compare NPM *targeted* to the hybrid protocol SCP.

To compare protocol efficiency, we consider three common metrics: throughput, delay and energy. We characterize the energy efficiency of the protocols by analyzing the contribution of signaling and synchronization to total energy consumption. For all protocols, the length of the signals and the number of signals required to send all data messages together determine the total signaling overhead. For NPM, these two metrics also indicate the effectiveness of adaptive preambling and neighborhood wakeup respectively.

While the underlying protocols determine the time a radio spends in each radio state (transmit, receive, idle and sleep), the power consumed at each state is determined by the radio characteristics. For our evaluation, we used the power profile of a MICAz radio [41] (current draw in transmit mode = 17.4mA, receive mode = 18.8mA, idle mode = 20uA, and sleep mode = 0.1uA) with an external power source of 3.0V.

We evaluated protocol behavior across different traffic patterns, and in networks with different neighborhood sizes using *ns-2*. The target network contained 100 nodes in a $10\times$ 10 grid network. To analyze the effect of neighborhood size on performance, we varied the spacing among the nodes from 200m to 140m to 75m. With a radio coverage range of 250m, these node distances created a neighborhood of size 4, 8 and 18 nodes respectively. Since we obtained similar trends for the different node densities, only results for 140m are presented.

All protocols except IEEE 802.11 PSM and SCP were evaluated in randomly synchronized networks (i.e., random and unsynchronized wakeup times). SCP and IEEE 802.11 PSM were always run with synchronized wakeup schedules. Moreover, the results for IEEE 802.11 PSM in this section, does not capture any control overhead due to its costly [38] out-of-band synchronization mechanism and thus only acts as a baseline in an ideal scenario. The only control overhead that we consider for PSM is the synchronization overhead due to beaconing.

Nodes in NPM, B-MAC and SCP duty cycle using a 1ms awake duration in every 100 ms period. Nodes in X-MAC remained awake for 2ms in every 100ms. The higher duty cycle for X-MAC was to support receiving the strobed preambles. Both NPM and PSM used a 100ms ATIM window and a 600ms data window for its nodes. Experiments were performed with various window sizes, and showed similar trends. To make the simulations more realistic, all protocols (except IEEE 802.11 PSM) added random clock drifts (maximum drift 100 ppm = 100s drift per 1 million s) to each node. To handle the drift, in our simulations, NPM and SCP added a 1ms guard time to its preambles and refreshed its neighbor tables every 60s. For NPM, the thresholds that we used for selecting the preambling nodes are: queue len = 20 packets, age of the

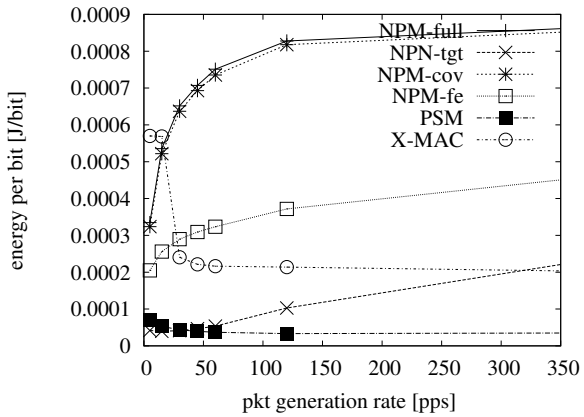

Figure 2.7: Energy per bit: CBR traffic

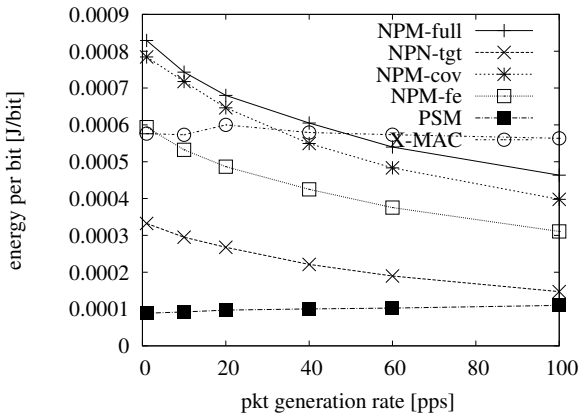

Figure 2.8: Energy per bit: Bursty traffic

oldest packet = 4s, and the number of preambles sent = 20 preambles. These parameter values for thresholds were obtained empirically via extensive simulations.

To analyze protocol performance in different traffic scenarios, we evaluated the protocols with two traffic patterns: CBR (common for habitat monitoring applications) and bursty (common for event detection applications). Since traffic bursts increase the opportunity to utilize neighborhood wakeup, it lays out a favorable scenario for NPM and X-MAC. The effectiveness of neighborhood wakeup in CBR traffic scenario depends on the traffic generation rate. To make the CBR traffic more realistic, we chose the inter-arrival time between packets for each node from an exponential distribution. To create the different load conditions in this pseudo-CBR scenario, we varied the exponential average from a 600sec packet inter-arrival time for low load to a 5sec inter-arrival time for high load. For the bursty traffic, we simulated the protocols in a network, where 9 nodes at one corner of the 100 node grid generated bursts every 60s. We increased the number of packets in each burst to overload the network. In both cases, all data packets (size 80 bytes) were destined to a sink (transmit data rate of 250 Kbps [41]), which was placed in the distant corner of the grid. All protocols used shortest hop routing.

All results are obtained from an average of 10 simulation runs. For each run, all protocols were simulated for 1 hour.

2.4.1 Energy

To compare the energy profiles of the protocols at different rates, we use *energy per bit* as a metric. This metric captures the contributions of both data transmission and control overhead to the total energy con-

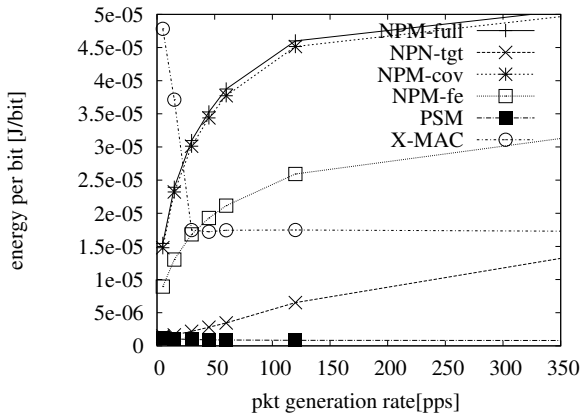

Figure 2.9: Signaling overhead per data bit: CBR traffic

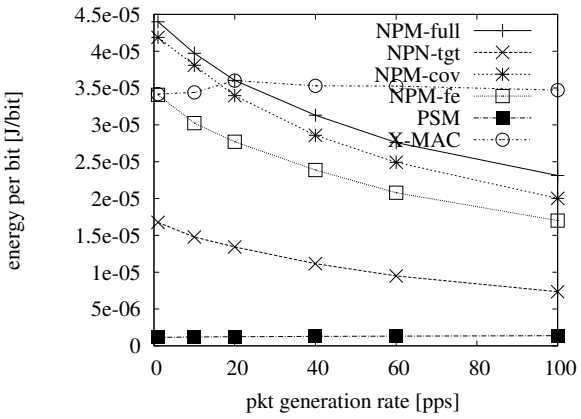

Figure 2.10: Signaling overhead per data bit: Bursty traffic

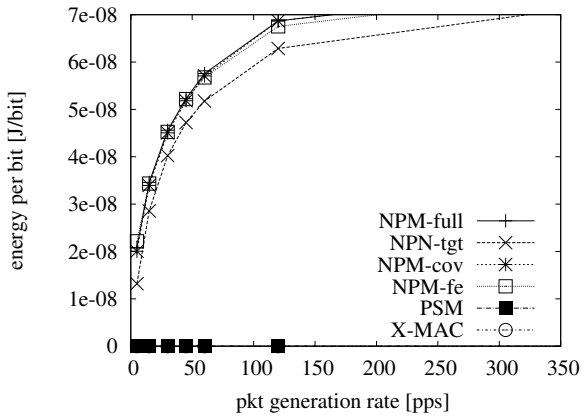

Figure 2.11: Synchronization overhead per data bit: CBR traffic

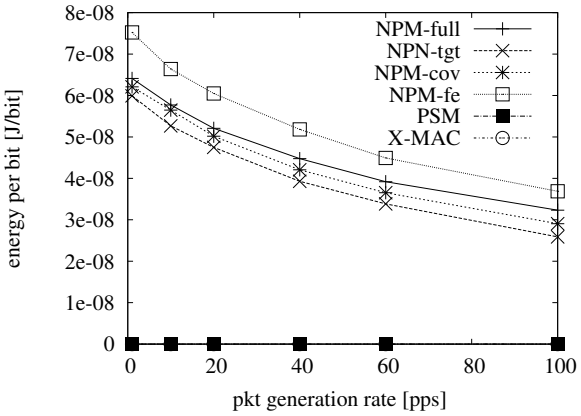

Figure 2.12: Synchronization overhead per data bit: Bursty traffic

sumption. Thus, any energy-efficient protocol that incurs a low control overhead, will have a low energy per bit value.

As the traffic rate increases, the total energy per bit for each protocol follows either of the two interesting trends (see figures 2.7 and 2.8). For both CBR and bursty traffic, energy for the protocols that allow neighborhood wakeup, such as PSM and NPM, either remain stable or follow a decreasing trend with higher rates. For the rest of the protocols, B-MAC, X-MAC and SCP, energy consumption increases as the rate increases. To understand these trends, we analyzed the signaling (see figures 2.9 and 2.10) and the synchronization overhead (see figures 2.11 and 2.12) incurred by each of these protocols.

Signaling overhead is the only component of B-MAC and X-MAC's control overhead. Due to the one preamble per packet policy, both of these protocols consume fixed signaling overhead per bit at low

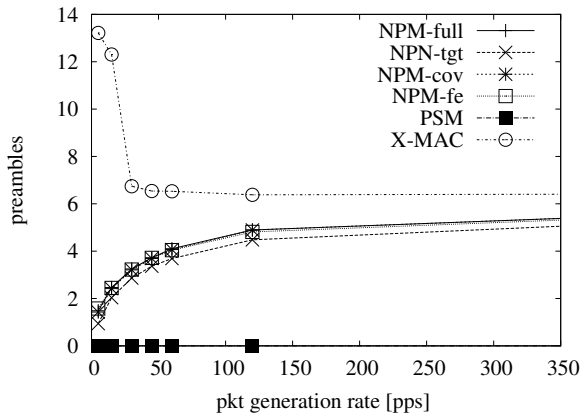

Figure 2.13: Preamble count per data: CBR traffic

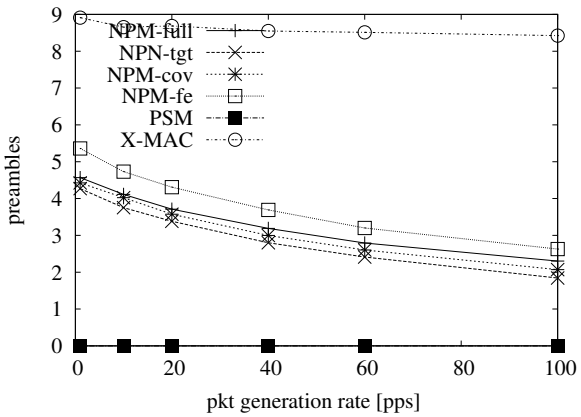

Figure 2.14: Preamble count per data: Bursty traffic

rates. However, as the network gets overloaded, more packets get dropped due to collision and contention, increasing the energy consumed per received data bit (we omit B-MAC from the total energy per bit graphs due to its very high and skewed values relative to the other MAC protocols). The strobed preambles of X-MAC allow it to avoid overhearing and to have shorter preambles, resulting in much lower signaling overhead than B-MAC. For the same reason, the increasing trend of X-MAC starts at a much higher traffic rate than it does for B-MAC.

Although NPM uses a combination of both signaling and synchronization mechanisms, it is the signaling overhead that shapes the total energy per bit of NPM. Since NPM amortizes the cost of both signaling and synchronization over multiple transmissions (see figures 2.13 and 2.14 for the number of preambles required per data message), its total energy follows a decreasing trend as the traffic rate increases. The huge energy efficiency of NPM *full* compared to its dual B-MAC for all traffic, shows the effectiveness of neighborhood wakeup in reducing total energy. However, the efficiency of NPM *finish early* to its dual X-MAC is dependent on both the type and the rate of traffic. NPM *finish early* consumes less energy than X-MAC only when the effectiveness of NPM's neighborhood wakeup outperforms the benefit of X-MAC's very low signaling overhead. Achieving this goal becomes even more difficult at extremely low rates. Thus, despite the neighborhood wakeup, NPM *finish early* consumes higher energy per bit for CBR traffic until the rate becomes sufficiently high. NPM *targeted*, however, by using very low overhead for signaling, consumes much less energy than X-MAC for almost all traffic rates except for extremely low rates. At extremely low rates, NPM *targeted* has very few opportunities to use its neighborhood wakeup and to use the targeted preambles, causing targeted to consume higher energy than X-MAC. The situation

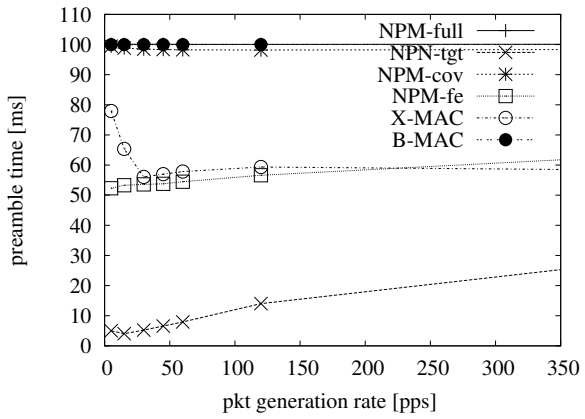

Figure 2.15: Average preamble length: CBR traffic

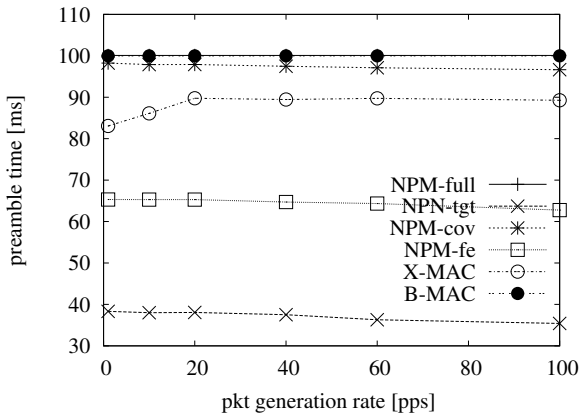

Figure 2.16: Average preamble length: Bursty traffic

is completely different with bursty traffic. Despite the traffic burst support of X-MAC, X-MAC uses its preambles less efficiently than NPM *finish early*. Due to NPM's broader traffic support, NPM *finish early* uses fewer preambles than X-MAC (see figure 2.13 for CBR and figure 2.14 for bursty traffic). This allows NPM *finish early* to have a steeper decreasing trend for signaling overhead per bit than X-MAC. Thus, NPM *finish early* outperforms X-MAC even at very low burst rates.

SCP's low total energy per bit stems from the very low signaling overhead it incurs. SCP achieves this by spending more energy in synchronization. At low rates, SCP does better than NPM *targeted* approach. But, as the rate increases, NPM's neighborhood wakeup amortizes the cost of both of its signaling and synchronization over multiple transmissions, causing NPM *targeted* to consume much less energy than SCP. However, for the bursty traffic, NPM *targeted* always perform better than SCP due to the effective neighborhood wakeup.

For NPM, at high rates, all approaches can effectively utilize the entire DATA window. So, the benefits of opportunistic sending becomes less important, and reducing the average length becomes the main source of energy-efficiency. During the simulations, for full, coverage, finish early and targeted, NPM's average preamble length is 100ms, 80ms, 50ms and 5ms respectively (see figure 2.15 for CBR and figure 2.16 for bursty traffic). Thus, NPM *finish early* and NPM *coverage* incur lower signaling overhead than NPM *full*. NPM *targeted*, having extremely short preambles, incurs the lowest signaling overhead among all NPM approaches at all traffic rates for both CBR and bursty traffic.

NPM *targeted*, which is the best in terms of energy among all NPM approaches, reaches very close to PSM in a perfectly synchronized network (we consider no control overhead for PSM due to explicit

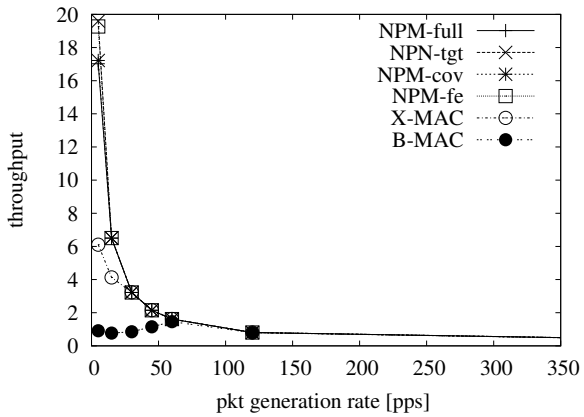

Figure 2.17: Throughput: CBR traffic

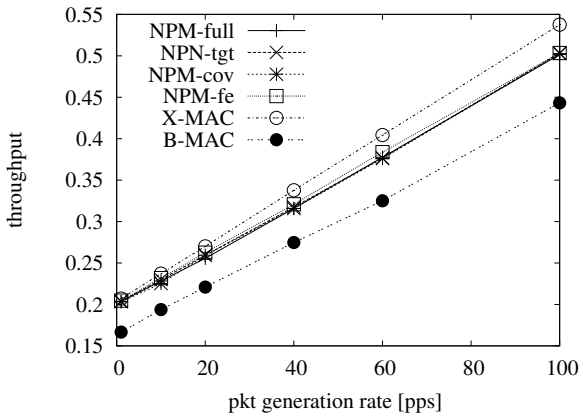

Figure 2.18: Throughput: Bursty traffic

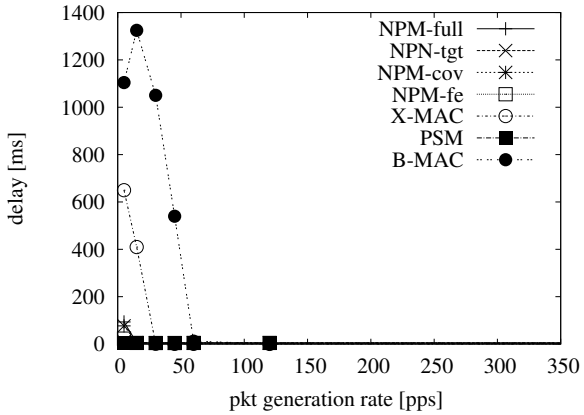

Figure 2.19: Average Delay: CBR traffic

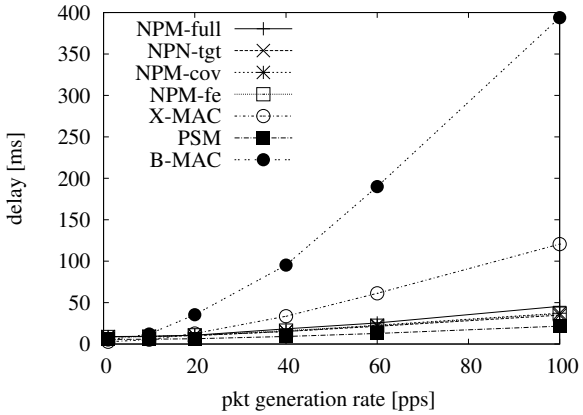

Figure 2.20: Average Delay: Bursty traffic

synchronization). For higher rates, NPM *targeted* performs even better than PSM.

2.4.2 Delay and Throughput

Both the neighborhood wakeup and the adaptive preambling mechanisms play important roles in shaping the delay characteristic of the NPM approaches. During low traffic, NPM and PSM, the two protocols that allow neighborhood wakeup, incur higher delay than B-MAC and X-MAC, due to the introduction of a 100 ms control and a 600 ms data window. NPM delays the packets even more because of the preambles. SCP, on the other hand, in spite of using very short preambles, incurs higher delay than PSM and NPM *full*, due to the high contention from synchronized wakeups. The delay becomes worse for SCP for traffic bursts (see figure 2.20). At low traffic, X-MAC incurs the lowest delay due to its relatively shorter preambles than B-MAC.

Although the neighborhood wakeup mechanism induces higher delay in NPM and PSM at low traffic, it allows these protocols to maintain a relatively steady delay (more than 90% reduced delay of NPM *targeted* at high CBR rate) and a linear throughput (see Figure 2.17, 2.18) at higher traffic. B-MAC and X-MAC's inability to handle higher traffic causes the network to overload at a relatively low rate, CBR rate of 1 packet per 120sec for B-MAC, and CBR rate of 30sec for X-MAC (see Figures 2.19, 2.20). For NPM, the neighborhood wakeup determines the delay performance at high traffic; however, once the network gets saturated, delay is entirely determined by the effectiveness of the adaptive preambling component, causing NPM *targeted* to have the lowest delay at relatively higher traffic. At high burst rates, NPM drops slightly more packets than X-MAC, since there is more contention within the transmission phase. However, NPM throughput is always within 5% of X-MAC's throughput (even at the highest rate).

When looking at all of the results, NPM *targeted* is extremely efficient in terms of energy but suffers from higher delays for bursty or low rate CBR traffic. In these scenarios, when delay is the main concern, NPM *finish early* provides good performance at a reasonable energy cost.

2.5 Conclusions

The contribution of this work is the design of *Neighborhood-based Power Management (NPM)* [14, 11, 10], an energy-efficient MAC protocol that integrates opportunistic sending and opportunistic scheduling to balance synchronization and signaling overhead. The novelty of NPM lies in its usage of the wakeup signals. Unlike other signal-based approaches, NPM uses these signals as neighborhood wakeup tones, enabling communication for all nodes in the neighborhood of the signaling node and amortizing the cost of signaling over multiple transmissions. Moreover, NPM uses opportunistically gained knowledge about the sleep schedules to adapt and ultimately reduce the signal length (i.e., preamble) based on one or a set of destinations. Finally, the combination of the two mechanisms effectively, NPM reduces the energy consumption across a large variety of network conditions: bursty traffic and CBR with high traffic rates. In dense networks, the neighborhood wakeup awakes more nodes, hence the cost of signaling is amortized over more neighbors. Moreover, the frequent traffic increases opportunities to obtain full knowledge about the wakeup schedules with data-driven synchronization and resulting in low synchronization overhead. Hence, by leveraging the characteristics of dense networks, NPM enables energy-efficient coordination that supports efficient use of the communication channel with improved communication delay.

Chapter 3

Bankrupting Jammers in Dense Networks

The well-defined communication in duty-cycled networks can expose information about wakeup schedules and traffic patterns and facilitate jammers to formulate energy-efficient jamming attacks. The effect is detrimental in WSN where such jammers can significantly reduce the network lifetime. In our research, we show that, high density allows the neighbors opportunities to collectively defend against the jammer by obscuring these information. Without such information, the entire neighborhood appears as a single entity to the jammer and prevents the jammer from achieving selective jamming. As the density increases, the jammer needs to attack more nodes to be effective, hence reducing its own energy-efficiency. We propose `Jam-Buster`—a jam-resistant solution for energy-constrained wireless networks, orthogonal to the existing anti-jamming solutions, that increases resilience by using multi-block payloads, eliminates differentiation of packets by using equal size packets and reduces predictability of traffic patterns by randomizing the wakeup times of the sensors. While each of these individual components of `Jam-Buster` is quite simple, the combination of the three components results in a jam-resilient system that forces the jammer to transmit more enabling faster detection of the jammer by the nodes, and to spend more energy to be effective and so reduce its own lifetime.

3.1 Energy-efficient Jamming in WSN

The combined effect of the broadcast nature of the wireless medium and the limited energy resources available to the sensor nodes makes WSNs extremely vulnerable to jamming attacks. By interfering with the ongoing transmissions, the jamming signals not only reduce the system throughput by causing packet drops, but also attack the network lifetime by wasting both the senders' and the receivers' energy. Since most existing protocols are designed only for energy-efficiency without considering the presence of a jammer in the system, the jammer can waste a significant amount of the sensors' energy using very few short transmis-

sions, making detection difficult and spending only a little of the jammer's energy [88]. Such an asymmetry of cost enables almost undetectable jammers with limited energy to launch successful attacks against a wireless sensor network.

Current solutions to defend WSNs against such cheap jammers are all based on hiding ongoing packet transmissions from reactive jammers. Such jammers must first detect a packet before they can transmit a signal that jams the packet. If a reactive jammer cannot detect a packet, the jammer cannot jam that packet. To achieve this, the sender can try to hide the packets using either a randomized Start-of-Frame Delimiter (SFD) [84] for each packet, channel surfing [87] or a combination of both [84]. However, in both of these techniques, senders and receivers must be tightly synchronized to successfully transmit packets. This results in high overhead even when there is no jammer or the jammer is jamming with a low probability. Moreover, if the jammer has multiple radios and can listen to multiple channels at the same time, or if there are multiple jammers in the neighborhood each listening to a different channel, channel surfing can no longer hide packets from the jammer. Overall, while these techniques may be able to hide some packets from the jammer with similar hardware capabilities, these solutions are quite expensive and have no defense against cheap jamming when the packets are detected.

Several factors contribute to the effectiveness of such cheap yet hard to detect jamming in sensor networks. First, due to the lack of effective error correcting codes, packets have very low *resilience* to interference. Essentially, a jammer only needs to disrupt a few bytes inside a packet to effectively jam the entire packet. The cost for transmitting these few bytes by the jammer are several orders of magnitude lower than the cost of the sender transmitting and the receiver receiving the entire packet. Second, the easy *differentiability* of data and control packets facilitates selective jamming attacks causing the jammer to incur low transmission cost, and yet achieve the same effectiveness as jamming almost all packets. Third, with current periodic MAC protocols, a statistical jammer can easily *predict* future data transmission times. This allows the jammer to duty cycle to avoid unnecessary idle listening costs while still detecting and jamming all packets. In this section, we discuss the current solutions to jamming attacks in light of these characteristics. Finally, we discuss how we approach these problems as a whole in `Jam-Buster`.

3.1.1 Resilience

Standard communications in WSNs use direct sequence spread spectrum (DSSS) [67] with offset-quadrature phase shift keying (OQPSK) to modulate their signals [41]. Although DSSS protects communication against random noise in the channel, it fails to provide any resilience against intentional jamming since the pseudo-noise (PN) code used by the DSSS in WSN is also known to the malicious nodes. Moreover, due to lack of any packet level protection against errors, the jammer needs to jam only one symbol inside a packet to effectively jam the entire packet. In wireless networks, error correction codes (ECC) [52] provide this packet level protection against errors. However, the low data rate of a WSN channel, the short packet length and the limited resources of sensors make traditional ECC [52, 48, 79, 39] for wireless networks, infeasible for WSN. Hence, a feasible ECC must be simple yet efficient. Moreover, it must be adaptive and probabilistic so that a smart jammer can not force the sensors to always use a high level of redundancy in their packets and thus exhaust their energy.

Instead of recovering the entire packet, we propose to recover parts of the packet using a low overhead, low complexity mechanism when the packet is partially jammed. By improving resilience, we can increase the cost of the jammer's transmissions. We will show that by enabling the amount of that detection to be adapted dynamically, the cost of the jammer can be increased to the point where the jammer's lifetime is significantly reduced while still allowing the sender to recover a significant portion of the transmitted data.

3.1.2 Differentiability

By observing the length and interarrival time between packets, a jammer can easily distinguish the type of a packet, even if the packet content is encrypted. Typically, control packets are shorter than data packets and have smaller inter-packet spacing. In WSN, the reactive jammer knows the packet length even before the radio finishes the packet transmission since the `CC2420` radio transceiver attaches the packet length information at the beginning of every packet. Given knowledge of the message type and the specific protocol being used, a jammer can target to prevent the transmission of data packets in the first place by CTS corruption jamming, or to force the sender to retransmit the packet multiple times until the sender gives up and drops the packet from its queue by applying DATA and ACK corruption jamming [81].

By removing differentiability, knowledge of the specific MAC protocol no longer benefits the jammer, which now must decide to jam packets without knowing the type of the packet. We will show that by

eliminating the correlation between packet type and inter-frame spacing as well as the correlation between packet size and packet type, the jammer's cost is again significant increased.

3.1.3 Predictability

By observing the temporal arrangement of packets induced by the nature of the MAC protocol, a jammer can easily identify a pattern in the data transmission times. Once the pattern is detected, an energy-efficient jammer wakes up only during the predicted data transmission times and thus saves idle listening costs by sleeping during the rest of the time. Obviously, the pattern detection algorithm [46] used by the jammer will be different for different MAC protocols since the temporal organization of medium accesses varies with MAC protocols. However, the patterns are always created either due to the periodic sending of the senders or the periodic listening of the receivers, both of which are determined by the periodic duty-cycle of the nodes in the network.

The key to reducing this vulnerability is to break any existing pattern between the successive wakeup times. A similar concept is implemented at the Los Angeles International Airport since August 2007 to maximize the security of checkpoints given the limited security resources [62, 63]. Game theory was used to determine a random schedule for patrolling that breaks any existing patterns in selective patrolling, making the schedules difficult to predict for the attackers. We will show that WSN can also benefit from randomization to defend against jamming. An intelligent MAC protocol can use random wakeup times to effectively cause the jammer to either stay awake all of the time or to miss a significant number of packets.

3.1.4 Road to Jam-Buster

Instead of trying to outsmart the jammer, we take a completely different approach that attacks the jammer's detectability and energy, and is also orthogonal to the existing solutions. Essentially, our goal is to make the jammer spend more time awake and more time transmitting to successfully jam its target packets, increasing the jammer's detection vulnerability and its energy consumption. The contribution of our research is `Jam-Buster`, a low overhead jam-resistant framework that attacks the three factors that contribute to cheap jamming—resilience, differentiability and predictability.

3.2 Jam-Buster

`Jam-Buster` is a jam resilient framework for MAC protocols that provides improved performance and reduced energy consumption in the face of a reactive jammer, while causing the jammer to significantly increase the number of jamming signal transmissions facilitating jammer detection and so limit its impact, lifetime or both. First, `Jam-Buster` uses a *multi-block payload* that improves resilience by enabling the receiver to independently check different blocks inside a packet. As a result, a jammer's disruption is limited to those blocks with which the jamming signal overlaps. Hence, `Jam-Buster` forces a jammer to jam more bytes to successfully jam a whole packet. Second, `Jam-Buster` eliminates differentiability by using the same size packets for both data and control packets, and by randomizing the inter-spacing between the packets. Without differentiability, a jammer does not know the type of a given packet and so is forced to jam all packets that it detects, as opposed to cheaper approaches, such as jamming only control messages. Third, `Jam-Buster` reduces predictability by getting rid of the periodic pattern of wakeup times. Nodes wakeup randomly following a pseudo-random number generator, resulting in random data transmission times and forcing a jammer to remain awake all of the time to detect and jam all packets.

While each component of `Jam-Buster` is quite simple and contributes to increasing the cost of jamming, any component alone only has limited impact on the jammer. For example, resilience only increases the length of the actual jamming signal for packets that the jammer decides to jam. This alone is not effective for detection, since a jammer may only need to jam a few well-chosen packets, or for energy, since the jammer may have good predictability and be able to sleep between transmissions. Similarly, good solutions for both resilience and predictability fail if the jammer has good differentiation. The real benefits come from the combination of the three components, which results in a jammer that must jam all packets using longer jamming signals, incurring high transmission and listening costs as well as forcing the jammer to transmit more often, exposing itself to detection.

In the next three sections, we discuss how `Jam-Buster` tackles these problems using a *multi-block payload* to improve resilience, *look-alike packets* to eliminate differentiability and *random wakeup* to eliminate predictability.

3.2.1 Multi-Block Payload: Defense against Low Resilience

`Jam-Buster` provides a level of protection inside each packet that is of low cost to the sender but of high cost to the jammer to jam. Instead of using a single CRC at the end of each packet like traditional 802.15.4 packets, the sender includes additional checks throughout the packet. The benefits of the multiple checks is twofold. First, these checks help the receiver to determine if there is any valid data in the packet. Second, these checks force the jammer to jam more bytes within each packet to cause the entire packet to be discarded.

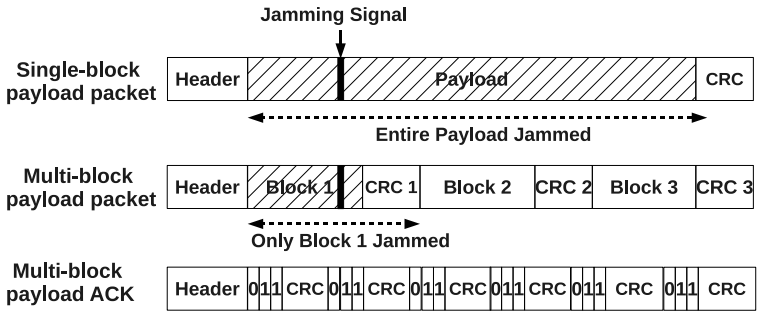

Figure 3.1: Packet format of a multi-block payload packet

`Jam-Buster` divides the payload inside each packet into multiple blocks where each block has its own CRC. We refer to this packet structure as a *Multi-Block Payload Packet*. Figure 3.1 shows the structure of a 3-block payload packet. For a reactive jammer to jam a packet, the jammer needs to switch its radio to the transmission mode, after detecting the Start-of-Frame-Delimiter (SFD) of the ongoing transmission. In our experiments as described in Section 4.3, we observed that due to the turn around time to switch to transmission mode, the reactive jammer could never jam the header of the ongoing packet. As a result, we have chosen not to repeat the header for each block. However, if this were not the case, we would need to repeat certain header fields, in each block.

The per-block CRCs enable the receiver to independently verify each block for corruption. After receiving the packet, the receiver marks a block inside the packet to be *jammed* only if the CRC for that particular block fails and marks the block as *unjammed* otherwise. In Figure 3.1, we marked the jammed portion of a packet as hatched areas. We observe from this example that the short jamming signal jams the entire payload of the traditional 802.15.4 packet, whereas it can only jam the first block of the 3-block payload packet. The data bytes in the rest of the blocks remain unjammed.

Since the receiver can successfully receive some of the blocks inside the packet when the jamming

signal only partially interferes with the packet, the traditional acknowledgement indicating the receipt of the entire packet is no longer adequate. Rather, the receiver must specifically acknowledge the receipt of each successfully received block inside the packet. Hence, the receiver uses a single acknowledge packet where a bitmap represents the success or failure of the reception of successive blocks inside the packet. Figure 3.1 shows the structure of an acknowledgement packet to a 3-block payload data packet. The $\{0, 1, 1\}$ bitmap of the ACK response indicates to the sender that the receiver successfully received block 2 and 3, whereas failed to successfully decode block 1. Hence, the selective ACK allows the sender to selectively retransmit the blocks that were jammed.

For multi-block payloads to be successful, it is extremely important to choose the right number of blocks inside the packet. With more blocks, *multi-block payload* provides finer grained protection from jamming. However, as the number of blocks increases, the overhead due to CRC also increases. Since WSN packets are small, with more blocks, the CRC overhead could overwhelm any benefit achieved due to multi-block payload packets. In Section 3.3, we discuss in detail how `Jam-Buster` chooses the optimal number of blocks in its packets.

3.2.2 Look-alike Packets: Defense against Differentiability

Packet size and inter-frame spacing are two key indicators of different packet types. `Jam-Buster` uses two techniques to make all packets *look-alike*. First, all packets are the same size, regardless of the amount of information or the packet type. Second, inter-frame spacing is randomized to prevent the jammers from using communication patterns to determine packet type. To complete the solution, all packets are encrypted to hide the type information inside the packet header.

Generally, control packets are shorter and data packets are longer. `Jam-Buster` has two options to make the control and the data packet sizes equal: either break each long packet into multiple short packets, or pack each short packet inside a long packet. Since each packet needs its own header, using multiple short packets incurs an increased header overhead. In contrast, the overhead due to longer packets incur because the control packets do not have sufficient information to fill up the inflated payload. However, with *multi-block payload packet*, discussed in Section 3.2.1, the sender can intelligently use the extra space to improve the jam-resilience of its control packets. In Figure 3.1, we provide an example of applying a combination of both *multi-block payload* and *look-alike packets* to an ACK packet. The same block is repeated multiple

times to fill up the entire payload to provide high jam-resilience as well as to eliminate differentiability. Due to this increased protection, Jam-Buster chooses to use longer packets for both its control and data packets. The only downfall of this approach is the overhead paid by the long control packets when there is no jammer present in the network. But this overhead is not significant in a WSN environment since packets in a typical sensor networks are really small. Even the maximum payload size in WSN is limited to 80 bytes. However, we understand that the overhead will be significant in a 802.11 network since 802.11 data packets are large compared to the control packets. If Jam-Buster was to be used for 802.11 networks, then such overheads could be reduced by using cumulative ACKs or by eliminating the need for ACKs altogether by using erasure coding. However, all these mechanisms are out of the scope of this paper, and we are only interested in Jam-Buster's applicability in a WSN scenario. When a jammer is present in the network, the overhead due to long packets becomes minimal compared to the benefits achieved from more jam-resilient control packets.

Jam-Buster uses a random inter-frame spacing, RIFS, between each of its packets belonging to a single transmission. If $\texttt{IFS}_{min}$ is the minimum turnaround time required to switch the radio from the reception to the transmission state and $\texttt{IFS}_{max}$ is the maximum allowable inter-frame spacing (i.e., $\texttt{IFS}_{max} = \texttt{EIFS}$), the value of RIFS is chosen from the following random number generator:

$$\texttt{RIFS} = \texttt{random } (\texttt{IFS}_{min}, \texttt{IFS}_{max}).$$

This implies that when a receiver receives a packet, it waits for a randomly chosen RIFS before sending its ACK. On the other hand, since the sender has no knowledge about the RIFS chosen by the receiver, the sender waits $\texttt{IFS}_{max}$ before finally considering the ongoing transmission to be unsuccessful. In addition to that, the sender must also wait for at least an $\texttt{IFS}_{max}$ duration before starting its own transmission in order to make sure the previous transmission has been completed. Hence, $\texttt{DIFS} > \texttt{IFS}_{max}$. The overhead of using RIFS is a slightly increased delay due to the longer waits between packet transmissions. However, our model in Section 3.3 shows that the increase is minimal and Jam-Buster benefits from *look-alike packets* due to more successful transmissions resulting in lower overall delay.

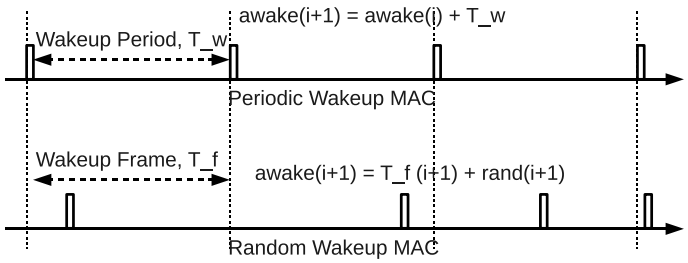

Figure 3.2: Random wakeup

3.2.3 Random Wakeup: Defense against High Predictability

Existing MAC protocols have periodic wakeup schedules to save energy. An attacker can listen to determine the period of a node's wakeup schedule, and then can jam that one node and still sleep just as much as that node. To create asymmetry between the jammer and a legitimate network user, `Jam-Buster` randomizes the wakeup time of each sender, making it unpredictable to the jammer.

Each node divides its time into equal sized slots called *wakeup frames*, T_f (see Figure 3.2). These wakeup frames are analogous to the wakeup periods in a periodic MAC protocol. However, with random wakeups, a node wakes up at a random time within each wakeup frame. The offsets r_n from the start of the wakeup frames are determined from a pseudo-random number generator `PRNG` - the seed of which is unique for each node. The combined effect of the unsynchronized start of the wakeup frames for different nodes and the different offsets for the different wakeup frames results in a unique wakeup pattern for each node in the network.

Using random wakeups, `Jam-Buster` eliminates predictability in two ways. First, due to the random r_n in each slot, the jammer can no longer predict future wakeup times of a node by correlating the temporal arrangements of the previous wakeup times of the same node. Second, because each node has a unique seed, the jammer can no longer predict the future behavior of a node by watching the behavior of another node in the same network. In this situation, the only way for the jammer to predict a node's future wakeup times is to have exact knowledge of the wakeup parameters of that node, i.e., the seed of that node's `PRNG`, the length of T_f and the clock offset between the jammer and the sensor node, $\texttt{offset}_{S-J}$. Given this information, the jammer can determine node A's current wakeup frame number, $\hat{n} = \lceil \frac{t+\texttt{offset}_{S-J}}{T_f} \rceil$, at

time t and then the exact wakeup time, $\hat{a_n}$, in that wakeup frame:

$$\hat{a_n} = (\hat{n} - 1).T_f + (r_{\hat{n}} \ \texttt{mod} \ T_f) - \texttt{offset}_{S-J}.$$

However, these parameters are exchanged in encrypted packets, the key to which is not known to the jammer. Hence, we can safely assume that the jammer cannot obtain the wakeup parameters and hence cannot generate the predicted wakeup times.

Since the jammer can no longer predict the future transmission times with random wakeups, the only way for a reactive jammer to detect and jam all packets is to remain awake all of the time. However, such an *always-on jammer* will have a very short lifetime due to its high idle listening costs, and hence will not be very effective. Our model in Section 3.3 validates this claim. The other option for the jammer is to duty cycle and be effective for a longer time, but fail to detect any transmission that occurs while it is asleep. Thus, the jammer has to make a tradeoff. The jammer must choose between a higher duty cycle to increase the number of packet detections and hence successful jamming, and a lower one to save energy and extend the jammer's lifetime. In Section 3.3, we will show that in a network with random wakeups, the *duty-cycled* approach is dominated by an *always-on* strategy since the *duty-cycled* strategy results in more missed opportunities for the jammer to jam packets.

3.2.4 Coordination with Random Wakeup

The biggest challenge of using random wakeups is to coordinate the sender-receiver pairs to ensure successful data transmissions. An ideal coordination mechanism must be *jam-resilient* as well as *energy-efficient*. If the coordination itself is not jam-resilient, the jammer can exploit this vulnerability to pose energy-efficient jamming attacks, essentially nullifying the benefits obtained due to random wakeups. In this section, we analyze the existing MAC protocols to identify which protocols are appropriate to coordinate the nodes in a system using `Jam-Buster`.

The simplest way to achieve coordination is to use any existing random wakeup MAC protocols, e.g., RAW [61], PW-MAC [78]. In both protocols, coordination is achieved by transmitting beacons at the beginning of each wakeup. This incur extremely high overhead at low traffic rates due to the non-data driven beacon transmissions. As a result, such protocols are not appropriate for the diverse traffic expected in wireless sensor networks for detecting events. However, if the network expects high traffic, `Jam-Buster`

can adopt any of these protocols .

For diverse traffic, `Jam-Buster` can choose an appropriate periodic MAC protocol and adapt it to support random wakeups. Typically, hybrid protocols, such as NPM [10] and Wise-MAC [30], are the most energy-efficient solutions to coordinate nodes with periodic wakeups and diverse traffic. These protocols use a combination of *signaling* and *synchronization* to balance the signaling cost at high traffic and the synchronization overhead at low traffic. While signaling allows a sender to communicate with its receiver when the receiver's recent wakeup information is not available, synchronization helps to keep the signal lengths shorter by utilizing estimates about the receiver's wakeup times. To coordinate nodes with random wakeups using these periodic hybrid protocols, the *base* protocols require different modifications for their signaling and the synchronization components.

We can derive the required modifications from the two major differences that exist between periodic and random wakeups. First, the duration between two successive wakeups of a node with periodic wakeup is always T_f, whereas with random wakeup, this duration can vary between $(0, 2T_f)$. Hence, the sender may need to preamble for a maximum duration of $2T_f$ to make sure it successfully signals the receiver. Moreover, the sender must send short preambles during this duration instead of sending full length preambles to prevent the jammers from having perfect predictability due to the long preambles. Second, only T_f is no longer enough to predict future wakeup times of the receiver. With random wakeups, the sender must have additional information about the seed of the receiver's `PRNG`, and the clock offset between the sender S and the receiver A, $\texttt{offset}_{A-S}$. To reduce the effect of clock drift on the timing information, each node sends its next slot number n and the offset to the start of its n-th wakeup frame, $\texttt{offset}_n$. If S receives this information at time t_1, S can easily determine node A's slot number, $m = n + \lfloor \frac{t_2 - t_1 - \texttt{offset}_n}{T_f} \rfloor$ and from that the wakeup time, a_m at any given time t_2:

$$a_m = (t_1 + \texttt{offset}_n) + (m - n).T_f + (r_m \ \texttt{mod} \ T_f).$$

Hence, to enable random wakeup, the *base* protocol must be modified such that each node includes the additional wakeup information inside its synchronization packets, and the sender sends its data packets at the calculated wakeup time, a_m. To provide protection against the jammer, all information must be encrypted inside the packet. This implies modifying the ACK packets of Wise-MAC [30] and the CTRL packets of NPM [10] to include random wakeup information.

Variable	Meaning	Values
k	number of blocks	varying
ℓ_{jam}	jamming signal	varying
L	maximum payload	80 B
ℓ_{crc}	CRC length	6 B
ℓ_{data}	data bytes per block	
b_{jammed}	jammed blocks	
$\texttt{DATA}_{\texttt{unjammed}}$	unjammed bytes	
T_f	wakeup frame	1 msec
T_a	awake interval	100 msec
$\texttt{d}_{jammer}$	jammer's duty cycle	varying
$\texttt{d}_{RN}$	node's duty cycle	1%
T_d	packet inter-arrival time	varying
P_i	idle power	1.278 uJ
P_t	transmission power	52.2 uJ
P_r	receive power	59.1 uJ
x	relative battery level	varying
$\texttt{rate}_c$	channel rate	250 Kbps

Table 3.1: List of notations for the Jam-Buster analytic model

In summary, Jam-Buster is not restricted to using any specific MAC protocol for coordinating its sender-receiver pairs. Essentially, Jam-Buster can use any existing random wakeup protocol in case of a high traffic scenario. In case of a low traffic or diverse traffic scenario, Jam-Buster can be layered on any appropriate periodic MAC protocol; that is, one that uses short preambles and estimates the receiver's wakeup time to shorten its preamble duration.

3.3 Analytic Model of Jam-Buster

We model the interaction between a legitimate user that uses Jam-Buster and an intelligent jammer to find out an upper bound on the best performance of the jammer. The model essentially validates Jam-Buster's feasibility by showing that the upper bound lowers significantly when the regular nodes use Jam-Buster as their defense. Table 3.1 lists the notations used to explain the model. Moreover, the analysis also provides guidelines on the optimal number of blocks to be used by the sensors and the optimal strategy to be used by the jammer in terms of choosing its duty cycle and jamming signal length.

3.3.1 Interdependence between Strategies

The regular nodes and the jammer have two completely opposite goals. While `Jam-Buster`'s primary goal is to maximize the total number of bytes successfully received at the receiver, $\texttt{DATA}_{\texttt{unjammed}}$, the jammer's goal is to minimize the same by jamming the packets. $\texttt{DATA}_{\texttt{unjammed}}$ essentially depends on both the number of blocks, k, used by the sender in that *multi-block payload* packet and the length of the jamming signal, ℓ_{jam}. While an increased k benefits the sender by reducing the average number of jammed blocks in a packet, $b_{jammed} = \texttt{min}(\lceil \frac{\ell_{jam}}{\ell_{data}+\ell_{crc}} \rceil + 1, k)$, larger k also has the negative effect of reducing the number of data bytes inside each block, $\ell_{data} = \lfloor \frac{L}{k} \rfloor - \ell_{crc}$. This results in $\texttt{DATA}_{\texttt{unjammed}}$ to be extremely sensitive to the choice of k.

$$\texttt{DATA}_{\texttt{unjammed}} = (k - b_{jammed}).\ell_{data}.$$

Choosing the optimal k becomes even more complicated as $\texttt{DATA}_{\texttt{unjammed}}$ depends on b_{jammed}, which in turn depends on ℓ_{jam}. Moreover, there is no way for the regular node to obtain pre-knowledge about the ℓ_{jam} signal to be used by the jammer. An intelligent jammer can exploit this fact and change its ℓ_{jam} signals such that the sender can no longer maximize its $\texttt{DATA}_{\texttt{unjammed}}$ by using an optimal k all of the time. Since a fixed strategy is no longer adequate, we model the entire interaction using a game theoretic approach. We determine a mixed strategy for choosing k and ℓ_{jam} such that neither the sender nor the jammer can improve its payoff by changing the strategy. The $\texttt{DATA}_{\texttt{unjammed}}$ at this Mixed Strategy Nash Equilibrium (MSNE) state essentially determines the upper bound of the jammer's performance.

3.3.2 Modeling Jam-Buster using Game Theory

The game has two parties: a regular node, RN, that uses `Jam-Buster` and an intelligent jammer, J. Formally, we define the game as (S, f), where S is the set of strategy profiles (i.e., $S = S_{RN} \times S_J$) and f is the set of payoffs (i.e., $f = f_{RN} \times f_J$) of both the parties. Here, the sender's strategy S_{RN} is to choose k, whereas the jammer's strategy S_J is to choose ℓ_{jam}.

Payoff Functions

The sensor's payoff f_{RN} is determined by the total unjammed data bytes that the node receives during its lifetime. While $\texttt{DATA}_{\texttt{unjammed}}$ captures the total unjammed data bytes from a particular packet, it does not

capture the relative energy-constraint of the jammer and the regular nodes in the network. Since the jammer can only jam until its own lifetime expires, we need to incorporate the energy constraints into f_{RN}. Finally, the jammer's goal is to minimize f_{RN}. Hence, the interaction between the jammer and the nodes can be represented as a zero-sum game, and the jammer's payoff is defined as: $f_J = -f_{RN}$.

f_{RN} is a function of $\texttt{DATA}_{\texttt{unjammed}}$ and the relative lifetime, $\texttt{life}$, which in turn depends on the energy spent by the nodes and the jammer, E_{RN} and E_J respectively, and the relative battery level, x, of the jammer and the nodes:

$$\texttt{life} = \texttt{max}(\frac{E_j}{x.E_{RN}}, 1).$$

Both E_{RN} and E_J depend on the traffic scenario, the network density and the adopted duty cycles. We assume a single-hop network with n regular nodes and 1 jammer where each sensor generates traffic at T_d inter-arrival times and the jammer's target is to jam these transmissions. We limit our analysis to a single-hop network because the jammer can only jam those transmissions that occur within its one hop distance. All nodes wakeup randomly maintaining a duty cycle of d_{RN}. Hence, we determine E_{RN} within one T_d period as follows:

$$E_{RN} = \frac{L}{\texttt{rate}_c} P_t + T_d.\texttt{d}_{RN}.P_i.$$

We assume that jammer joins the network with a battery x times larger than that of a legitimate node. In our model, we keep x as a parameter and vary x from 1 to 100. This allows us to simulate the effect of multiple jammers present in the network with just 1 jammer. Essentially, we can consider the scenario with 2 jammers with each one having $\frac{x}{2}$ times larger battery than a legitimate node using the same single jammer scenario. We can also simulate jammers with unlimited energy resources with a very large x value. We assume that the jammer's duty cycle is $\texttt{d}_{jammer}$. In our model, an *always-on jammer* is represented with $\texttt{d}_{jammer} = 100\%$, while a no jammer situation is represented with $\texttt{d}_{jammer} = 0\%$. A jammer with $\texttt{d}_{jammer}$ duty cycle, can detect only $\texttt{d}_{jammer}$ portion of all packets. Hence, the jammer spends $(n.\texttt{d}_{jammer}.\ell_{jam}/\texttt{rate}_c)$ time transmitting to jam all $n.\texttt{d}jammer$ packets detected from n nodes. We calculate E_J as follows:

$$E_J(\ell_{jam}) = \frac{n.\texttt{d}_{jammer}.\ell_{jam}}{\texttt{rate}_c} P_t + T_d.\texttt{d}_{jammer}.P_i.$$

Finally, we incorporate both $\texttt{life}$ and $\texttt{DATA}_{\texttt{unjammed}}$ to obtain f_{RN}. Since, the jammer can detect and

jam a packet with $(1 - \mathtt{d}_{jammer}/\mathtt{life})$ probability, f_{RN} is as follows:

$$f_{RN} = \frac{\mathtt{d}_{jammer}}{\mathtt{life}}\mathtt{DATA}_{\mathtt{unjammed}} + (1 - \frac{\mathtt{d}_{jammer}}{\mathtt{life}})(L - k\ell_{crc}).$$

Here, we can see that when the jammer has unlimited energy resources, i.e., $\mathtt{life} = 1$, and the jammer does not duty cycle, i.e., $\mathtt{d}_{jammer} = 100\%$, the node's average payoff is simply the unjammed bytes from a packet, $\mathtt{DATA}_{\mathtt{unjammed}}$.

Solving the MSNE

To solve the MSNE, we calculate f_{RN} and f_J for all (k, ℓ_{jam}) values at different $\mathtt{d}_{jammer}$ settings, and then input these values to our MSNE solver written in `matlab`. The solution assigns probabilities to each strategy inside S_{RN} and S_J. If MSNE assigns probability p_i to the i-th strategy of S_{RN}, then `Jam-Buster` follows that strategy with probability p_i. The same is the case for the jammer.

The parameter values used in our `matlab` simulations are listed in Table 3.1. We varied the traffic by choosing different $T_d = \{5, 10, 60, 120\}$ `sec` values and the network density by setting $n = \{1, 2, 3, 4\}$. We changed the jammer's setting by setting $x = \{1, 2, 5, 10, 25, 50, 75, 100\}$ and $\mathtt{d}_{jammer} = \{0\%, 25\%, 50\%, 75\%, 100\%\}$. Interestingly, for all the settings, the MSNE solution indicated that the nodes must always use different k values with equal probabilities and in response the jammer must always send the longest ℓ_{jam} to maximize their payoff at the equilibrium state. For very high relative battery levels ($x > 80$), MSNE suggests the nodes to use the higher k values with slightly increasing probability. However, the jammer's strategy remains the same.

Next, we need to determine the jammer's duty cycle level at the equilibrium. We observe that irrespective of the current k used inside the packet, f_{RN} is the minimum when the jammer is always-on, i.e., $\mathtt{d}_{jammer} = 100\%$. This is expected from the f_{RN} equation since the jammer can now detect and jam all packets. For high traffic rates, we observe the same payoffs for both the *always-on jammer* and the duty-cycled jammer (see Figure 3.3). However, the situation is not the same for low traffic. The jammer then spends more time idle listening to detect packets rather than actually jamming packets. Hence, as the traffic inter-arrival time increases, the sender achieve lower payoff with higher $\mathtt{d}_{jammer}$ jammers, and the lowest when the jammer is always awake. At this point, the sender achieves only a 50% payoff which maximizes the jammer's payoff. Thus, the MSNE suggests that regardless of the shorter lifetime, the jammer should always remain awake

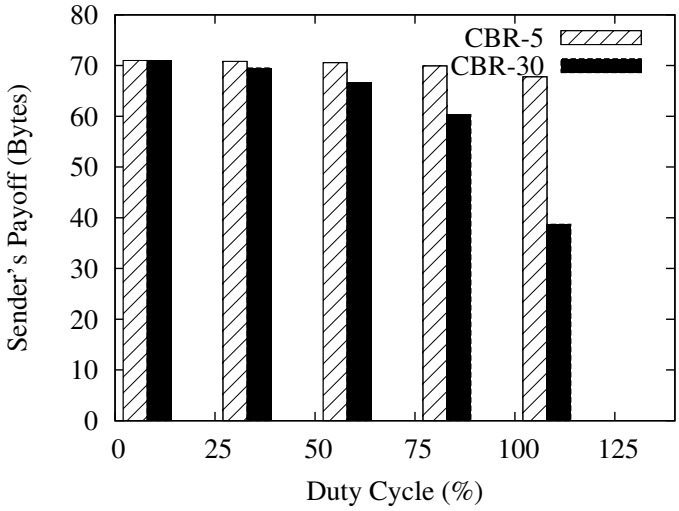

Figure 3.3: f_{RN} for different T_d from Jam-Buster analytic model

to achieve maximum payoff.

To summarize, our model shows that at the MSNE of the system, the jammer either transmits an excessive amount of jamming signals, resulting in higher detectability and a very short lifetime, or the jammer duty cycles resulting in longer lifetime but lower effectiveness. Our model shows that `Jam-Buster` reduces the overall efficiency of an intelligent jammer to an inefficient fixed-strategy always-on jammer that jams all packets.

3.4 Implementation Details

We implemented `Jam-Buster` in TinyOS, and ran the experiments on `Tmote Sky` motes.

3.4.1 Implementing `Jam-Buster`

For all sensors in the network, we implemented the three components of `Jam-Buster`. While implementing look-alike packets and random wakeups was easy, the multi-block payload required modifications to the radio transceiver and the `CC2420` driver of the `Tmote Sky` motes. The typical operation of a transceiver is to check the CRC of the packet and then pass the packet to the application layer only if the CRC passes. To enable the receiver to independently check each block, we disabled the default CRC enabling the `CC2420` transceiver to pass the packet to the upper layer regardless of whether it passes or fails the CRC. The appli-

cation layer then checks individual blocks inside a packet for interference.

3.4.2 Implementing the Reactive Jammer

A reactive jammer transmits its jamming signal upon detecting an ongoing transmission. In the default setting, the `CC2420` transceiver backs off its transmission when it finds a busy channel using a mechanism called clear channel assessment (CCA). While implementing our reactive jammer, to ensure that the jamming signal interferes with the ongoing packet, the jammer turns off its default CCA. Furthermore, the jammer must start transmitting very fast to ensure that the jamming signal coincides with the ongoing data transmission. We observed that transmitting a packet is a two-step process. At first, the `CC2420` transceiver loads the packet inside its `TXFIFO`. Then, the transceiver starts transmitting the packet onto the radio channel. From our experiments, we observed that loading a packet requires around $3-4$ `msec`. Within this time, the ongoing packet transmission is over. To avoid this delay due to loading time, we preload the `TXFIFO` with a jamming packet. The `CC2420` chip starts transmitting the packet as soon as it detects an ongoing transmission by calling the `CC2420Transmit.resend(FALSE)` function.

The reactive jammer listens to the channel for an SFD to detect an ongoing transmission. We implement the detection by attaching an interrupt to the SFD pin of the `CC2420` chip on the jammer. It is important to stop the SFD interrupt on the SFD of the jamming signal. We implemented two jammers: `jam-1` which sends 1 pulse when it detects a packet, and `jam-2` which sends 2 back-to-back jamming pulses to jam a packet. Due to hardware constraints, there is 11 bytes of unjammed data between the 2 pulses of `jam-2`.

3.5 Experimental Evaluation

The goal of our evaluation is to show that `Jam-Buster` can successfully improve the jam resilience of wireless sensor networks: (1) by forcing any jammer to transmit more jamming signals enabling faster jammer detection, and (2) by forcing an energy-constrained jammer to spend more energy jamming and thus bust the jammer. In both cases, the increased transmissions by the jammer facilitate the nodes faster detection of the jammer. Our evaluation also proves the feasibility of our proposed solutions by showing that `Jam-Buster` incurs only minimal overhead when there is no jammer in the system.

3.5.1 Evaluation Metrics

We evaluate Jam-Buster's effectiveness against two types of jammers: an *always-on jammer* and a *duty-cycled jammer*. We use the *average unjammed bytes per packet*, $\texttt{DATA}_{\texttt{unjammed}}$ to evaluate Jam-Buster's effectiveness. We also use *time-average unjammed bytes*, f_{RN} to capture the overall effectiveness of the senders and the jammer given their energy constraints. For both metrics, a high value represents better jam-resilience of the system, while a low value represents a more effective jammer.

For an energy-constraint jammer, we use *time-average unjammed bytes*, f_{RN} to capture the overall effectiveness of the senders and the jammer given their energy constraints. Similar to $\texttt{DATA}_{\texttt{unjammed}}$, a high f_{RN} indicates Jam-Buster's effectiveness while a low f_{RN} indicates the jammer's effectiveness. To capture the effect of relative battery levels of the jammer and the sender, x using our experiments, we evaluate *ratio of the battery exhaustion rate of the jammer to the sender*, life. When $x = 1$, life represents the relative lifetime of each sender compared to the jammer. A high life indicates the Jam-Buster's effectiveness in exhausting the jammer's energy resources.

3.5.2 Network Setup and Traffic Scenario

Our network is a single hop network with multiple senders sending CBR traffic to a sink, and one jammer. We varied the CBR inter-arrival times starting from 5 sec to 30 sec in steps of 5 sec. All senders wakeup randomly maintaining a 1% duty cycle. We vary the jammer's duty cycle from 100% to 0% in steps of 25%. While 100% duty cycle means that the jammer is always awake and thus can detect and jam all packets, 0% duty cycle means that the jammer is never awake representing a no jammer scenario. To capture the effect of jamming signal length, we evaluate our system with jam-1 and jam-2 jammers. In our evaluations, $k = 1$ represents the scenario when the nodes have no defense against the jammer, and k_{rand} represents Jam-Buster with nodes running *multi-block payload* with equal probabilities for all blocks, 1 to 7. Finally, k_{opt} represents the optimal scenario, where each node has the exact knowledge about the jammer's strategy, and thus can choose the optimal number of blocks for each packet to achieve maximum throughput. We want to emphasize that k_{opt} is not a feasible scenario with smart jammers, and is determined in this case using extensive experimentations. The results in this section are obtained from the average of 5 runs, where we ran each experimental setup for 30 minutes.

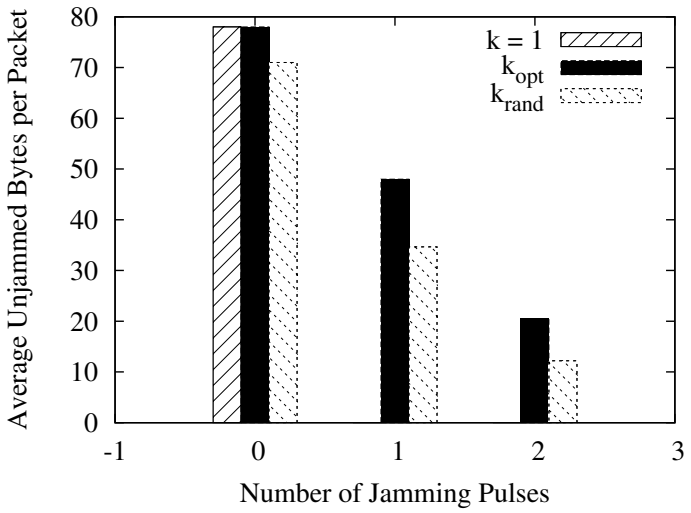

Figure 3.4: Unjammed bytes per packet with always-on jammer

3.5.3 Jam-Buster vs. Always-on Jammer

Our goal is to show that `Jam-Buster` improves jam-resilience against an always-on jammer by increasing $\texttt{DATA}_{\texttt{unjammed}}$ for the senders and thus by forcing the jammer to transmit more jamming signals to disrupt an entire packet.

With no defense against the jammer, packets are extremely vulnerable to jamming. Even with short jamming signals such as `jam-1`, the jammer can disrupt 100% of the packets (see Figure 3.4). However, when `Jam-Buster` is used, the receiver can successfully receive 44% of the packets. Even when the jammer increases its jamming signals and uses `jam-2`, the receiver can still retrieve 16% of the packet. By comparing $k = 1$ and k_{rand} for no jammer, we can see that `Jam-Buster` achieves this high resilience by incurring only 9% overhead due to additional CRCs. Essentially, this results in lower effectiveness of the jammer. With `jam-1` and `jam-2`, the jammer now can only jam 56% and 84% of the packets indicating that if the jammer wants to disrupt the entire packet, the jammer needs to send even more jamming pulses per packet. Such high transmissions from the jammer eventually leads to a very fast jammer detection.

3.5.4 Jam-Buster vs. Duty-cycled Jammers

Our goal is to show that `Jam-Buster` exhausts the jammer's energy at a rate such that the jammer is either very short-lived or not effective at all. All the results in this section are for a `jam-1` jammer.

Figure 3.5 shows the average unjammed bytes received while capturing the effect of the jammer's failure

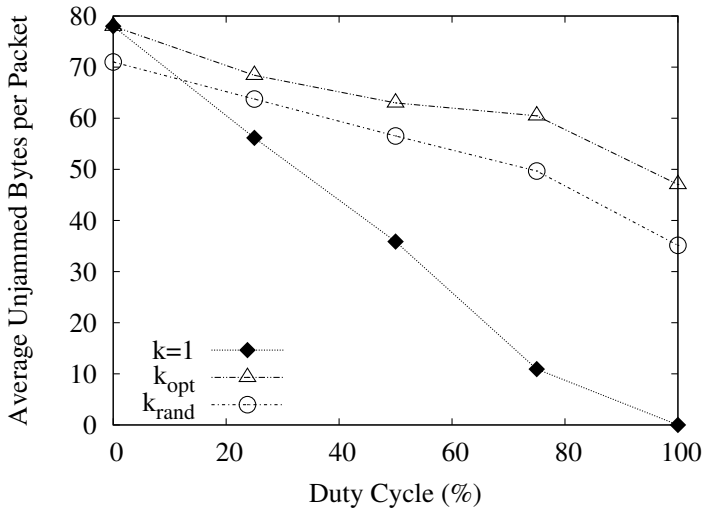

Figure 3.5: Unjammed bytes per packet with duty-cycled `jam-1` jammer

to detect packets when it is asleep due to duty cycling. Essentially, this represents f_{RN} in the model. The jammer's failure to detect and jam packets is obvious from the fact that the receiver successfully receives 72%, 46% and 14% of the packets when the jammer is following 75%, 50% and 25% duty cycles respectively (see Figure 3.5). With `Jam-Buster`, the receiver receives even more data: 82%, 72%, 64% and 45% of the packets respectively. Essentially, this indicates `Jam-Buster`'s success in eliminating predictability.

Essentially, the energy exhaustion ratio, `life` depends on the number of senders, n and CBR inter-arrival time, t_{CBR}. We observed a linear increase in `life` as percentage of duty cycle increases since the jammer spends more time idle listening. The high idle listening cost of the jammer dominates the impact of the increased transmission cost at high values of n. As a result, we did not observe any change in `life` for the different node density (Figure omitted due to space constraints). However, with lower traffic, we observed significant increase in `life`, a 275% increase from CBR-5 to CBR-30. As traffic becomes low, the jammer spends more of its battery looking for packets than actually jamming the packets.

To evaluate `Jam-Buster`'s effectiveness, we considered jammers with different battery levels, x. Figure 3.6 shows the time-average unjammed bytes for CBR-30 for different battery levels of the jammer. For both the cases, we observe that the duty cycle does not matter for $x = 1$. Nodes have the same time-average unjammed bytes. This is because of the relative lifetime. At low duty cycles, the jammer lives long but detects fewer packets during its lifetime. On the other hand, at high duty cycles, the jammer has a shorter lifetime, but detects more packets while it is alive. When averaged out, the jammer jams the same number

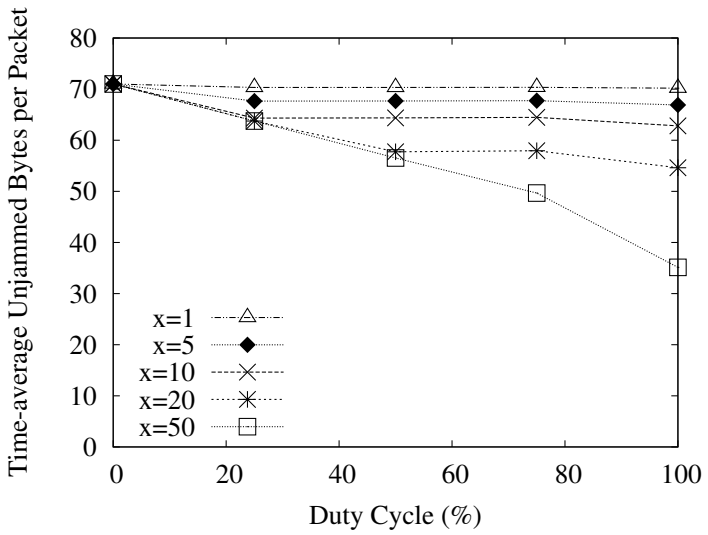

Figure 3.6: Time-average unjammed bytes per packet for CBR-30

of bytes. As a overall effect, the receiver receives data as if there was no jammer in the system. As x increases, the jammer lives longer, detecting more packets during its lifetime, thus reducing the time-average. However, the receiver still receives huge amount of data. Even with a 20 times larger battery, the sender still can receive 71% of the packets. With a jammer having 50 times more energy, the jammer is like an always-on jammer. We can see that the jammer has a trade-off. With 20 times battery, the jammer will do 50% duty cycle as it begets the same benefit as an always-on jammer, but has slower detection. For even more higher battery levels, if the jammer chooses to jam always, it becomes easily detected.

3.6 Conclusions

The contribution of this work is the design of `Jam-Buster` [13, 12], a novel approach to defend against the cheap yet undetectable jammer in WSN. By hiding the otherwise exposed schedule and traffic information, the entire neighborhood appears as a single entity to the jammer and prevents the jammer from achieving selective jamming. Essentially, the neighbors attack the jammer and force the jammer to transmit more jamming signals to achieve effective jamming. As the density increases, the jammer needs to attack more nodes to be effective. More transmissions from the jammer forces more energy consumption for the jammer and eventually results in faster detection of the jammer, essentially busting an energy-constraint jammer.

One application area of `Jam-Buster` is *detection networks in remote areas*, such as at border crossing [1, 2] or in poaching detection [4, 3] or in illegal logging detection. In these applications, sensor nodes

are deployed to monitor a large area for intruders and to alert the system if they detect any. To prevent detection, it is important for the intruders to carry jammers to jam these alert messages. However, the jamming signals themselves expose the jammer's location by causing increased packet loss. Hence, in this situation, it is only logical for the intruders to deploy a number of *jamming decoys*, for example mounted on wild animals, to confuse the monitor system of the actual intruder's location. When these decoys jam using sophisticated energy-saving techniques, they can be long-lived and substantially compromise the effectiveness of a remote monitoring system. Because both networks operate in remote areas, both normal users and jammers alike must carry their energy sources into the operating environment, and hence both face limited battery resources. Here, note that the jammers could use bigger batteries than the monitor nodes. However, in our approach, even with larger batteries, long-lived jamming cannot exist, forcing such adversaries to quickly exhaust their energy.

Chapter 4

Anycast to Reduce Delay in Dense Networks

Redundancy offers opportunities to improve the delay performance in duty-cycled networks. In dense networks, multiple nodes having similar routing metrics are often present within the transmission range of the sender. Essentially, the sender can choose any of these nodes to forward its data packets towards the sink. In an asynchronous duty-cycled network, these options provide an opportunity to the sender to reduce the sleep latency at each hop by forwarding its data packet to the next hop node that wakes up the soonest. Our research reveals that enabling MAC-layer anycast is a generic low-overhead technique. We propose AnyMAC, an extension that can be applied to any existing asynchronous MAC by using small modifications to enable anycast. Our evaluations show that the adapted protocols experience significant improvements in energy and delay.

4.1 Challenges for Reducing Delay

Delay in a duty-cycled network occurs when the sender waits for its next hop node to be awake. In a multi-hop scenario, the delay accumulates at each hop since each node along the path towards the destination must wait for its next hop node to be awake. For example (as seen in Figure 4.1), when sensor A transmits a data packet to the destination D along the path $A \rightarrow B \rightarrow C \rightarrow D$, delay accumulates as the nodes experience sleep latency at each of the three hops along the path towards the destination. The negative effect on delay performance deteriorates even more in dense networks as the traffic generation rate increases and the network gets saturated. Sleep latency is directly related to the period of the receiver's duty cycle. As the nodes sleep more and so save more energy, the sender must wait longer for the receiver to be awake [64, 89, 90]. The challenge to efficient communication in duty-cycled networks is to balance this energy-delay trade-off.

The most effective way to reduce the delay due to sleep latency is to carefully stagger the awake times

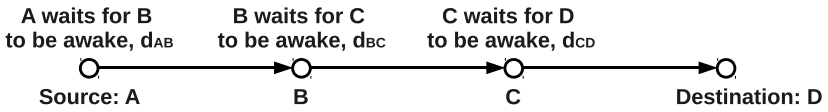

Figure 4.1: Delay accumulation at each hop due to sleep latency

of all nodes along the path towards the destination [51, 29, 74]. As a result, the sender at each hop no longer needs to wait for its next hop node to be awake. However, in a realistic network with unavoidable clock drifts and occasional path changes, the sensors end up expending excessive energy arranging the wakeup schedules of the next hop nodes, thus rendering such solutions to be in-efficient for large networks with diverse traffic patterns.

In response, anycast[1]-based solutions [61, 94, 95, 49, 77, 33] are proposed that do not synchronize the awake times at all, rather exploit the path redundancy (i.e., multiple next hop nodes) available in the network to opportunistically reduce the sleep latency. Typically, sensor nodes are densely deployed (with sensor density as high as 20 sensors per m^2 [27]) to ensure fault tolerance and reliability of communication. This leads to multiple nodes having similar routing metric to be present within the transmission range of the sender. Essentially, the sender can anycast i.e, choose any of these nodes to forward its data packets towards the sink. In a dense duty-cycled network, these multiple next hop nodes provide an opportunity to the sender at each hop to reduce the sleep latency by forwarding its data packet to the next hop node that wakes up the soonest.

The goal of any MAC-layer anycast protocol is to exploit the inherent redundancy in the network to improve the performance by allowing the sender to opportunistically forward its data packets to the best next hop node (according to some metric: e.g., reliability, packet delivery ratio (PDR), latency,distance etc.). MAC-layer Anycast [25] and ExOR [43] are the two early anycast MAC protocols designed to improve the reliability in a wireless adhoc network by opportunistically forwarding the data along the path having the best PDR. Latter, anycast protocols [61, 94, 95, 49, 77, 33] were proposed specifically to reduce the sleep latency in duty-cycled networks. In these protocols, at each hop, the sender opportunistically forwards its data packets to the best next hop node (according to some routing metric) that wakes up the soonest. The challenge for these protocols is to identify the best next hop node with minimal overhead. While some protocols periodically exchange information between neighbors to enable the sender to choose the best next

[1]By anycast we refer to MAC-layer anycast (i.e., anycast to the one hop neighbors), not end-to-end anycast.

hop node when necessary [61], other protocols actively probe the neighborhood before a data transmission and selects the best next hop node according to receivers response to the probes [94, 95, 49, 77, 33].

In synchronization-based MAC-layer anycast protocol (e.g.,RAW [61]), the nodes broadcast their schedule and location information (obtained from GPS) using beacon packets each time they wake up. Since the sender has all up-to-date information necessary to choose the best next hop node, the sender can easily choose the neighbor that wakes up the soonest and progresses the data towards the destination (i.e. has a better routing metric than the senders routing metric). However, the exchange of schedule information by the RAW nodes is not data-driven. Thus, RAW incurs very high synchronization overhead. At low traffic, this overhead may even dominate the total energy consumption making synchronization-based anycast protocols suitable for only high traffic scenario.

In probing-based MAC-layer anycast protocols (e.g., GeRaF [94, 95], CMAC [49], A2 -MAC [77], CBF [33]), the senders have no information about the wakeup schedules of their neighbors. Instead, the sender systematically probes the awake neighbors with RTS packets and then detects the best next hop node from the CTS replies from its awake neighbors. The GeRaF sender transmits different RTS packets for the different priority next hop nodes. The sender starts searching the subregion closest to the destination, and then proceeds to search the next subregion (which is second closest to the destination) if no awake node is detected in the first subregion. The search continues until either the sender detects an awake receiver in the current subregion or the sender has exhausted all of the subregions and detected no awake neighbors at all. In the end, GeRaF requires multiple RTS messages in multiple rounds to systematically search the different subregions to identify the current best next hop node (if there is any). The multiple RTS messages not only incur a high control overhead for GeRaF nodes, but the multiple rounds also take a long time to complete the best next hop node selection process in GeRaF. Moreover, the greedy geo-forwarding in GeRaF often end up taking a longer route or a more power consuming path than the optimal route, causing the nodes to have increased end-to-end delay and energy overhead. To reduce the number of control messages exchanged in the best next hop node selection process, CMAC probes the neighborhood with a single RTS packet. In CMAC, the order of the CTS replies from the awake neighbors reflects their corresponding priorities (according to some routing metric) as next hop forwarder. While the different priority neighbors reply at different CTS slots, the same priority neighbors reply at randomly chosen minislots within the same CTS slot. Although the reduced number of RTS packets lowers the control overhead of CMAC compared to

GeRaF, CMAC still requires a long time to complete its best next hop node selection process. The time actually depends on the number and the length of CTS slots and minislots. As the number of CTS slots and minislots increase, CMAC becomes more effective in determining the absolute best next hop node. However, more CTS slots and minislots reduce the efficiency of the protocol by increasing the entire time required to complete the best next hop node selection process. In response, A2 -MAC and CBF focuses on reducing the total time required to complete the best next hop node selection process. A2 -MAC and CBF achieve this goal by limiting the next hop nodes (according to some routing metric) that can reply to an RTS probe and thus can be chosen as the next hop forwarder. With the limited forwarder set, the sender chooses the first node that replies with a CTS to be the next hop forwarder. Although the A2 -MAC and CBF senders have faster best next hop node selection process, the fast selection process is achieved at the cost of high overhead for constructing and maintaining an optimal forwarding set [22]. Moreover, the limited number of nodes in the forwarding set curbs the opportunity of improving delay by exploiting redundancy. For all the probing-based anycast protocols, during the time the sender probes the channel to select the best next hop node, the sender essentially captures the channel. The other nodes must wait until they find a free channel. Due to this long capture time, delay increases. The delay aggravates as the traffic load in the network increases, rendering the probing- based protocols suitable only for low traffic scenario.

Given the diversity of traffic expected in wireless sensor networks, we take an alternative approach to reduce the sleep latency of a duty-cycled network using anycast. Instead of designing a new anycast-based solution for diverse traffic, we propose a novel anycast layer extension for existing MAC protocols. Our solution, Any-MAC, is a generic, simple and low overhead extension that can be applied to any asynchronous MAC protocol to enable anycast. Since Any-MAC decouples anycast from the underlying MAC protocol, Any-MAC can extend any MAC protocol without changing the protocol behavior. The benefit of the decoupled anycast layer is that: when Any-MAC extends a duty-cycled MAC protocol suitable for diverse traffic, then the Any-MAC extended MAC protocol is the “anycast-based MAC solution for diverse traffic” with an improved delay performance.

4.2 Any-MAC

Any-MAC is a generic extension which can be applied to any existing asynchronous MAC protocol (we refer to the protocol to be extended by Any-MAC as base protocol) to enable anycast. The extended protocol

preserves the base protocol behavior. Due to the simplicity and the generic nature of Any-MAC extension, Any-MAC can easily extend any asynchronous MAC protocol that is appropriate for a specific application and network scenario, and thus can provide a MAC protocol appropriate for that particular scenario with improved delay performance.

The basic idea of Any-MAC is simple. When a sender has multiple next hop options and these next hop nodes wake up at different times, the sender can reduce its sleep latency by forwarding its data packet to the next hop node that wakes up the soonest. We refer to this node as the best next hop node. Any-MAC determines the best next hop node according to the following two criteria (with decreasing priority):

- C1: The node that has the lowest routing cost.
- C2: The node that wakes up the soonest.

As the best next hop node, Any-MAC chooses the node with the lowest routing cost that wakes up the earliest (supporting lower cost is future works). If multiple nodes have the same routing cost, any of them can be chosen as the best next hop node. In this case, Any-MAC chooses one of the nodes randomly. Once the Any-MAC extension chooses the best next hop node, the base protocol considers the selected node as the receiver for that particular hop and then forwards the data packets towards this best next hop node.

4.2.1 Interaction with Routing Layer

In order to determine the best next hop node, the Any-MAC sender must know about its multiple next hop options. The routing protocol above the anycast layer provides the Any-MAC sender with this information. (e.g., node id, routing metric etc.) about its multiple next hop nodes. To avoid routing loops, the routing protocol enlists a neighbor N to be the next hop node of sender S only if N has a lower routing cost than the sender S and thus is capable of progressing the data towards the destination. In a very dense network, a sender can have many neighbors that can progress the data towards the destination. In such cases, Any-MAC enlists only k such eligible next hop nodes that are superior to the others according to some routing metric (e.g., latency, energy, PDR etc.). The reason for limiting the next hop nodes is to limit the contention in the network as well as to keep the overhead limited. When $k = 1$, Any-MAC extended MAC protocols work exactly like their base MAC protocol.

Our current Any-MAC extension considers the number of hops as the routing metric and enlists only k next hop options that have the best (and same) routing metric value.

4.2.2 Any-MAC Details

The goal of Any-MAC is to choose the best next hop node[2] among the multiple next hop options to be the forwarder. To define the problem formally, consider a scenario where the sender S has k next hop nodes $N_1, N_2, ..., N_k$. At a particular time t, these k next hop nodes wake up after $w_1, w_2, ..., w_k$ time units respectively. The goal of Any-MAC is to find the best next hop node $N_* = N_i$ of S at time t such that N_i wakes up the soonest (i.e., $w_i = \min(w_1, w_2, ..., w_k)$).

We propose two generic anycast extension techniques: one synchronization-based and one probing-based. The choice of the actual extension technique however depends on the base MAC protocol (the details on the MAC are discussed in section 4.2.3).

Synchronization-based Any-MAC Extension

The precondition for adopting this extension is that the sender knows the wakeup times of its neighbors. At time t, the sender knows that its k next hop nodes will be awake after $w_1, w_2, ..., w_k$ time units respectively. With wakeup information available, if the sender has data to send at time t, the sender waits for $(w_i - t)$ time (where $w_i = \min(w_1, w_2, ..., w_k)$), and then sends its data packets.

Probing-based Any-MAC Extension

The precondition for adopting the probing-based extension is that the sender has no knowledge about the wakeup times of its neighbors. The Any-MAC sender sends anycast probes, each of which enlists multiple next hop nodes inside the probe packet. Any next hop node that receives the packet will accept the packet and then contend for being the next hop node. To avoid ACK collisions from multiple receivers, the receivers of the anycast probe packet adopt a slotted ACK mechanism (see Figure 4.2). The time after the probe packet is considered to be divided into slots. Each slot is long enough to hold one ACK packet. Each receiver schedule their ACK transmissions in their respective ACK slots. Since we only consider next hop nodes with the same routing cost, all receivers have the same priority. Thus, the receivers choose their respective slots randomly. This slotted ACK mechanism also allows the sender to determine the next hop forwarder node in a distributed fashion. In this scheme, any receiver that detects a busy channel assumes that an ACK transmission from a higher priority next hop node is in progress and thus refrains from sending its ACK and

[2]By best next hop node, here we refer to the node that has the best routing metric among all the awake nodes.

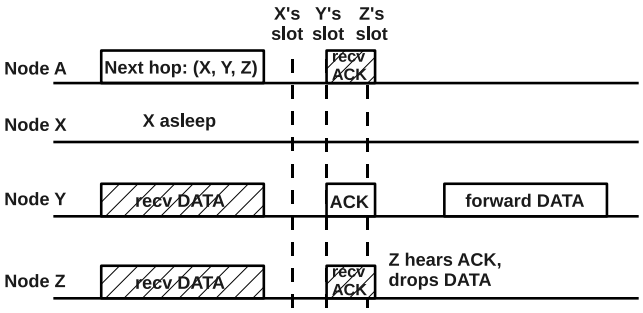

Figure 4.2: Slotted ACK mechanism for MAC-layer anycast

drops the data packet. Thus, the first next hop node that sends back the ACK will be the node that forwards the data packet.

4.2.3 Scope of Any-MAC

Any-MAC can successfully reduce the sleep latency in a duty-cycled network, given the network holds the following two conditions:

- A1: Different nodes wake up at different times.
- A2: The sender has multiple next hop options.

It is obvious that Any-MAC offers the most benefit when the wakeup times of the next hop nodes are completely random and each node along the path has multiple next hop options. However, these conditions are not strict. Any-MAC can still improve sleep latency as long as the next hop nodes are not completely synchronized and at least one forwarder along the path has more than one next hop options. Thus, the asynchronous MAC protocols [30, 75, 64, 20, 83, 10] for WSN are the appropriate candidates for exploiting redundancy to improve delay. The actual extension to enable anycast however depends on the mechanism that the MAC protocol uses to determine whether the next hop node is awake or not. Interestingly, these coordination mechanisms are analogous to the two types of anycast extensions: synchronization-based and probing-based.

Synchronization-based asynchronous MAC protocols [30, 75] periodically exchange schedule information so that the sender knows exactly when each of its neighbors will wake up. For example, WiseMAC [30] nodes broadcast their schedule message either using separate SYNC packets or by piggybacking their schedule information onto data messages. RI-MAC [75] nodes broadcast beacon messages at the beginning of

their wakeup times indicating to the sender that the next hop nodes are awake. Because of the readily available schedule information at the sender, these protocols can be easily extended using synchronization-based anycast extension. These synchronization-based protocols can take advantage of the route level redundancy. After the extension, the modified WiseMAC sender can easily choose the best next hop node by consulting its neighborhood schedule table. Similarly, for the anycast-enabled RI-MAC sender, the next hop node that sends the beacon message first is the best next hop node.

Probing-based asynchronous MAC protocols [64, 20, 83] are completely unaware of the wakeup schedules of their neighbors. These protocols mainly use two different mechanisms to identify their awake receivers. First, the senders transmit long preambles before their data messages (e.g., B-MAC [64]). The long preamble ensures that the next hop node detects channel activity when it wakes up and remains awake to receive data packets. Given multiple next hop nodes, B-MAC cannot detect the next hop node that wakes up the soonest since B-MAC always uses a fixed length preamble and its sleep latency does not depend on when the next hop node actually wakes up. Second, the senders probe the channel to detect whether its next hop node is awake or not. A reply from the awake next hop node indicates to the sender that the receiver is awake and ready to receive data packets. For example, X-MAC [20] and SpeckMAC [83] senders continuously probe the channel with repeated RTS and data packets respectively. Since the probing of these protocols is similar to the probing-based anycast extension, these protocols with short probes can be easily extended using the probing-based anycast extension. Thus the anycast-enabled X-MAC and SpeckMAC can detect the next hop node that wakes up the soonest by probing the channel for any of the next hop nodes instead of one.

Hybrid asynchronous protocols use both synchronization-based and probing-based mechanisms. For example, Neighborhood-based Power Management (NPM) [10, 14] uses a dynamic preamble to wake up its receiver and uses opportunistic synchronization (schedule information piggybacked onto control messages) to reduce the length of its preambles. However, NPM can reduce its preamble length only when recent schedule information is available. Additionally, NPM amortizes the cost of preambling over multiple data transmissions by enabling all nodes awakened by the preamble to send their data messages (referred to as opportunistic sending). Since the preamble may not wake up the receivers for all senders in the neighborhood (due to reasons such as the receiver being out of transmission range of the wakeup signaling node), NPM sender probes the channel with CTRL messages in order to identify whether the receivers are awake.

Since NPM uses both synchronization and probing, NPM can be extended using a combination of both synchronization-based and probing-based anycast extensions. Given multiple next hop nodes, NPM can benefit from the available redundancy in three different ways: First, multiple next hop options allow the wakeup signaling node of NPM opportunity to reduce its sleep latency by choosing the next hop node that wakes up the soonest as the target of the wakeup signal. Second, with multiple next hop options, NPM sender now has higher probability of having recent wakeup schedule information about at least one of the next hop nodes, thus increasing its opportunity to shorten the wakeup signal. Third, multiple next hop options increases the probability that a sender woken up by preamble will find an awake next hop node, thus increasing the possibility of opportunistic sending.

To summarize, asynchronous MAC protocols where the delay depends on the exact wakeup time of the next hop node can benefit from route level redundancy and diverse wakeup schedules of the nodes. Depending on the MAC protocol, redundancy can provide benefits in three different ways:

- Probing-based asynchronous protocols [20, 83] can benefit from redundancy since the sender can stop sending wakeup signals as soon as it detects any of the next hop nodes to be awake.
- Synchronization-based asynchronous protocols [30, 75] can benefit from multiple next hop options since the sender has the opportunity to choose the best awake next hop node using its knowledge about the neighbors' wakeup schedules.
- Multiple next hop options increases the probability of the sender finding an awake next hop node even when the sender sends its data opportunistically without sending a wakeup signal.

4.2.4 Example Any-MAC Extensions

It is relatively easy to augment appropriate MAC layer protocols. In order to benefit from redundancy, MAC protocols do not need to enable anycast for all of its messages (i.e., data and control). Once the sender identifies the next hop node that is awake with the help of anycast, the sender can directly send its data messages to that particular next hop node without performing anycast. Since MAC protocols generally identifies awake next hop nodes by using special control messages, we need to enable anycast for these specific control messages of the MAC protocols.

In this section, we discuss how Any-MAC can extend X-MAC [20] using the probing-based extension

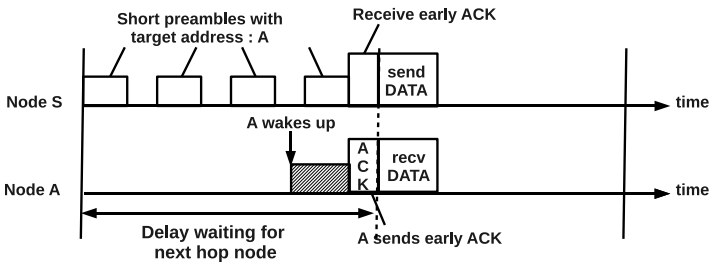

Figure 4.3: Original X-MAC protocol

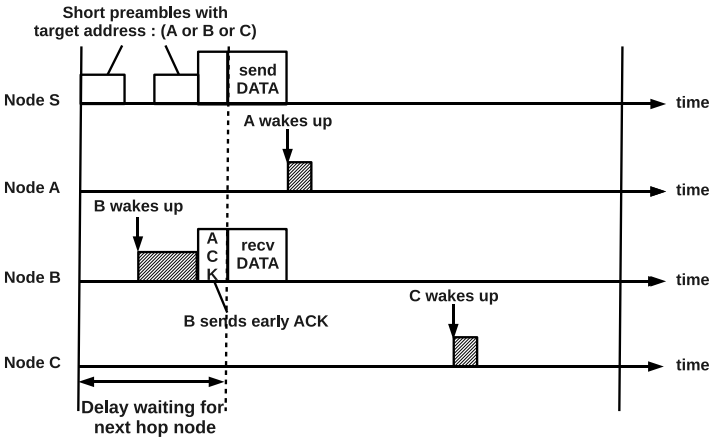

Figure 4.4: Any-MAC extended X-MAC protocol

and Neighborhood-based Power Management (NPM) [10, 14] using a combination of the synchronization-based and the probing-based extensions to enable these protocols to exploit redundancy in the network:

- The augmented X-MAC protocol sends `anycast wakeup signals` to enable the sender to identify the next hop node that wakes up the soonest. This next hop node sends back an acknowledgement according the slotted ACK mechanism. Once the awake next hop node is identified, the sender sends unicast data packets to that node.

- The augmented NPM protocol sends `anycast control messages` during its control window to identify whether any of its next hop options are awake. If a next hop node is awake, it sends back an acknowledgement using the slotted ACK mechanism. The NPM sender then directly sends its data messages to that particular next hop node during the following data window. We refer to this Any-MAC extended NPM as `npm-opp` in the evaluation section.

- Before sending the wakeup signal, the augmented NPM protocol consults the neighbor schedule table and identifies the next hop node that wakes up the soonest. The `anycast preambling node`

(i.e., the node that sends the wakeup signal) then selects that particular next hop node as the target of its wakeup signal, thus reaching the next hop node and then initiating the control window sooner. We refer to this Any-MAC extended NPM as `npm-sig` in the evaluation section.

The end result of this modification is improved wakeup signaling for X-MAC, and increased use of opportunistic sending as well as reduced wait time for NPM.

4.3 Evaluation

The goal of our evaluation is to analyze how MAC protocols for energy-constrained wireless networks can benefit from the inherent redundancy in these dense networks. As a proof-of-concept, we added Any-MAC anycast layer to two duty-cycled MAC protocols: X-MAC and NPM using *ns-2*, and analyzed their delay and energy performance while exploiting different levels of redundancy. We chose X-MAC as a candidate for the probing-based anycast extension, and NPM as a candidate for a combination of both the synchronization-based and the probing-based anycast extensions. Results for the augmented X-MAC with our proposed Any-MAC portrays how delay and energy can be reduced exploiting redundancy since anycast allows the sender to choose the first awakened next hop node. On the other hand, results for the augmented NPM with incorporated anycast layer Any-MAC shows how anycast improves its signaling and opportunistic sending (explained in section 4.2.3), resulting in improved delay performance.

4.3.1 Simulation Setup

We evaluated our prototypes using *ns-2* in two different network setups: an ideal setup and a realistic setup. In the ideal setup, each node along the path towards the destination has multiple next hop options and the wakeup times of the next hop nodes at each hop are equally spaced. Since, the ideal setup strictly satisfies A1 and A2 (as described in section 4.2.3), simulation results in this setup shows the maximum benefit achievable from Any-MAC anycast layer by exploiting redundancy. In contrast, in the realistic setup, all nodes along the path towards the destination may not have the same number of next hop nodes (some may even have 1 next hop option) and the wakeup times of the nodes are random (wakeup times of some next hop nodes may be clustered). Since, the realistic setup loosely satisfies A1 and A2, simulation results in this setup shows the average case benefit achievable from Any-MAC anycast layer.

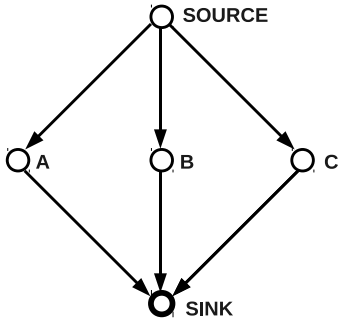

Figure 4.5: Redundancy in simple network (Ideal setup)

In our simulations, the ideal setup consists of 5 nodes with 1 source node, 1 sink node and 3 intermediate nodes acting as the next hop options for the source node. (Figure 4.5 shows the simple network used for our simulations). The wakeup times of the 3 next hop nodes are equally spaced to ensure maximum diversity of the wakeup schedules. Our simulation setup consists of 100 nodes in a grid setup with 1 sink node and 99 source nodes. The wakeup times of all nodes are chosen from a uniform random distribution. To make it more real, we capture the effect of clock drifts by introducing a clock drift of 100 `ppm` (parts per million) at each node. In our simulations, we space the nodes at 140 meter distance from each other. Thus, a node (with 250 meter transmission range) can have a maximum of 8 neighbors. We limit the maximum number of next hop nodes that a node can have is only limited to 3. This is because all neighbors of a node can not progress (have a lower routing metric) its data towards the sink. Essentially, the exact number of next hop nodes available to a node depends on the node's location in the grid and its distance to the sink. Figure 4.6 shows the number of next hop nodes available at each node for a 25 node grid network (for clarity of the figure, we use a 25 node network) with similar setup. Note that, in our simulations we use a larger network with 100 nodes.

Each node in the network (both ideal and realistic setup) wakes up periodically maintaining a 100 `msec` wakeup period. During each wakeup period, the X-MAC nodes remain awake for 2 `msec`, whereas the NPM nodes remain awake for 1 `msec`. The reason for the longer awake time of X-MAC lies in its signaling mechanism. Since X-MAC senders wait long enough to receive acknowledgement packets after their signals, the X-MAC receivers must listen to the channel for longer time while they are awake to detect signals directed towards themselves [11]. The other protocol specific parameters used in our simulations are listed in Table 4.1.

We used CBR traffic (typical for environment monitoring applications) to evaluate effectiveness of Any-

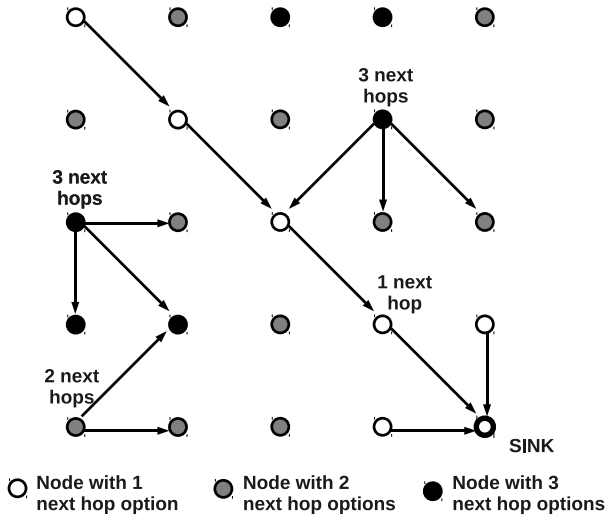

Figure 4.6: Redundancy in a 5×5 grid network (Realistic setup)

Control window	30 msec
Data window	600 msec
Guard time around wakeup signal	2 msec
Refresh timer for neighbor schedule table	60 msec
Immediate timer	10 msec

Table 4.1: NPM parameter values

MAC at different load conditions. By varying the inter-arrival time between successive CBR packets from 600 `sec` to 5 `sec`, we simulated the different load conditions in the network (we show only the results from 250 `sec` to 5 `sec` for better visual representation of the graphs). Thus, in our simulations, an inter-arrival time of 600 `sec` represents a low load scenario, whereas a 5 `sec` inter-arrival represents a high load scenario. Our simulations use shortest path routing protocol to forward the packets from a particular source to the sink. Thus, the forwarding set of an Any-MAC node consists of those next hop nodes that can route the data using the fewest hops. We vary the size of the forwarder set k from 1 to 2 to 3 to evaluate the effectiveness of the Any-MAC anycast layer given different levels of redundancy. With $k = 1$, the Any-MAC extended protocols act as their original version of the protocols that do not exploit redundancy.

4.3.2 Evaluation Metric

We compared the delay and energy performances of each baseline protocol, X-MAC and NPM, with its Any-MAC extended version, `xmac-k` and `npm-k` (here, k represents the size of the forwarding set, with

Voltage	3.0 V
Transmission power (P_{tx})	$17.4mW$
Reception power (P_{rx})	$18.8mW$
Idle power (P_i)	$1mW$
Sleep power (P_s)	$0.1mW$

Table 4.2: Power profile of CC2420

$k = 1$ representing the original baseline protocol without anycast). The comparison shows the effectiveness of Any-MAC in opportunistically improving the delay and energy performance of an asynchronous MAC protocol by exploiting route level redundancy.

We use *delay per hop* as the metric for comparing delay performances. Delay per hop captures the sleep latency of the sender at each hop, and thus indicates the effectiveness of the Any-MAC anycast layer. In addition to reflecting the sleep latency, delay per hop also captures the network's ability to handle high traffic scenario. To better reflect the ability of the network to handle network saturation, we used *delivery ratio* as an additional metric.

We use *energy per bit* as the metric for comparing the energy performance across different traffic generation rates. Energy per bit captures the control overhead as well as the data overhead incurred for transmitting one data bit, and thus indicates the efficiency of the Any-MAC anycast layer. In our simulations, we used the power characteristics of CC2420 radio transceiver to calculate the total energy consumption.

4.3.3 Results in an Ideal Setup

In the ideal setup, with 2 next hop nodes in the forwarder set (i.e., $k = 2$), Any-MAC extended X-MAC (`xmac-2`) observes a 34% reduction (see Figure 4.7) in average delay per hop compared to the original X-MAC protocol without anycast (`xmac-1`). When the size of the forwarder set is further increased to $k = 3$, `xmac-3` observes an additional 20% improvement in average delay per hop over `xmac-2`. Thus, more next hop options provide Any-MAC with higher opportunity to improve delay. However, the percentage of improvement diminishes as the number of next hop options increases (as seen in Figure 4.7). In the end, the delay trend for Any-MAC in an ideal setup closely tracks the expected n-fold improvement in sleep latency when the sender has n next hop options. In addition to improvement in delay, Any-MAC extended X-MAC also observes similar improvement in energy. The energy consumption reduces as Any-MAC reduces the number of wakeup signals to be sent by the sender in the process of reducing its delay.

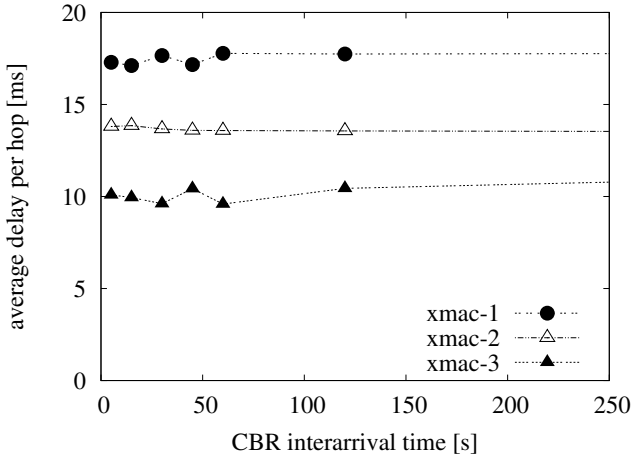

Figure 4.7: Delay for Any-MAC extended X-MAC: Ideal setup

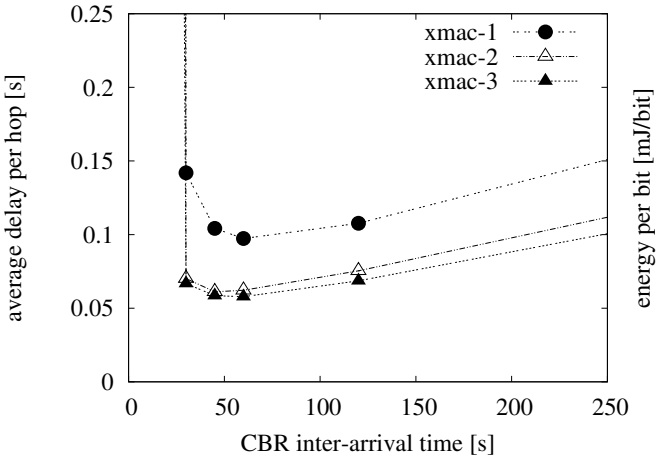

Figure 4.8: Delay for Any-MAC extended XMAC: Realistic setup

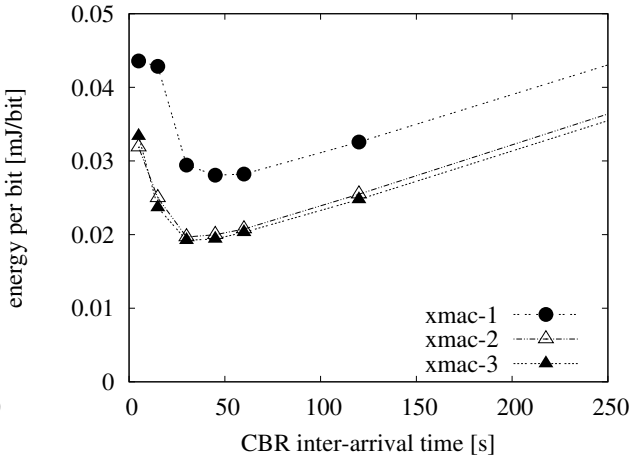

Figure 4.9: Energy for Any-MAC extended XMAC: Realistic setup

4.3.4 Results in a Realistic Setup

In the realistic setup, different senders can have different maximum numbers of next hop options, m (ranging from 1 to 3) depending upon its position on the grid. As we vary the size of the forwarder set $k = (1, 2, 3)$, a sender with m maximum next hop nodes can effectively take advantage of $\texttt{min}(k, m)$ next hop nodes.

Results for Any-MAC extended X-MAC

With 2 next hop nodes in the forwarder set, `xmac-2` achieves a 33% improvement (see Figure 4.8) in average delay per hop compared to `xmac-1`. However, as we further increase the size of the forwarder

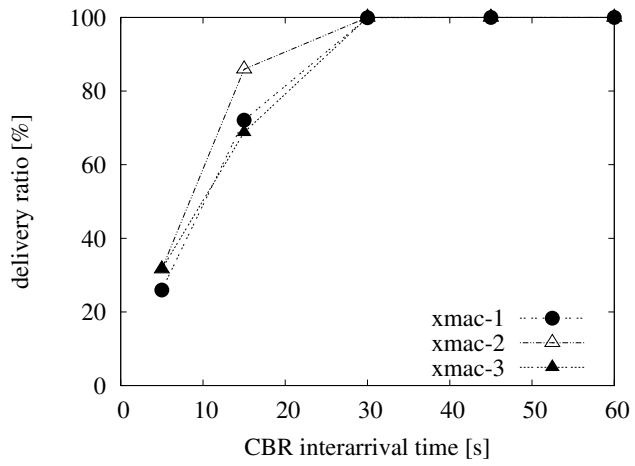

Figure 4.10: Delivery ratio for Any-MAC extended X-MAC: Realistic setup

set ($k = 3$), `xmac-3` observes only an 8% additional improvement over `xmac-2`. The lower percentage improvement of `xmac-3` over `xmac-2` in the realistic setup compared to the ideal setup arises from the fact that in the realistic setup, not all senders in the network have 3 possible next hop nodes. However, almost all senders in the network have at least 2 possible next hop nodes (as illustrated in Figure 4.6). Thus, with a forwarder set of size $k = 2$, the senders can almost always take advantage of the 2 next hop nodes. In contrast, with a forwarder set of size $k = 3$, the senders can rarely take advantage of all 3 next hop nodes. Thus, `xmac-3` shows only marginal improvement over `xmac-2`. By reducing the the number of wakeup signals, Any-MAC extended X-MAC also achieves similar energy improvement (see Figure 4.9) as delay, with the increased number of next hop nodes.

An added advantage of the reduced delay in Any-MAC extended X-MAC is that the extended protocol can better handle high traffic. As the size of the forwarder set k increases, Any-MAC allows the senders to send their data packets faster through the network so that queues only build up at a relatively higher traffic rate. Thus, Any-MAC extended X-MAC enjoys a better delivery ratio (see Figure 4.10) even at high traffic scenario.

Results for Any-MAC extended NPM

As explained in Section 4.2.4, we implemented two Any-MAC extensions to NPM. First, `npm-opp` enables anycast for the control messages of NPM using probing-based anycast extension so that the extended NPM can gain benefits from increased possibility of opportunistic sending. Second, `npm-opp+sig` enables

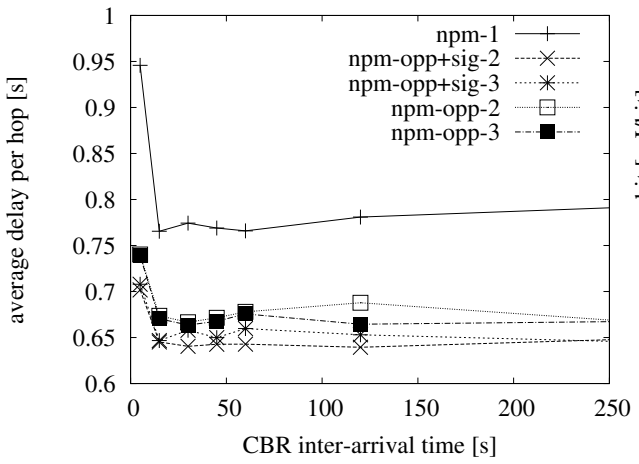

Figure 4.11: Delay for Any-MAC extended NPM: Realistic setup

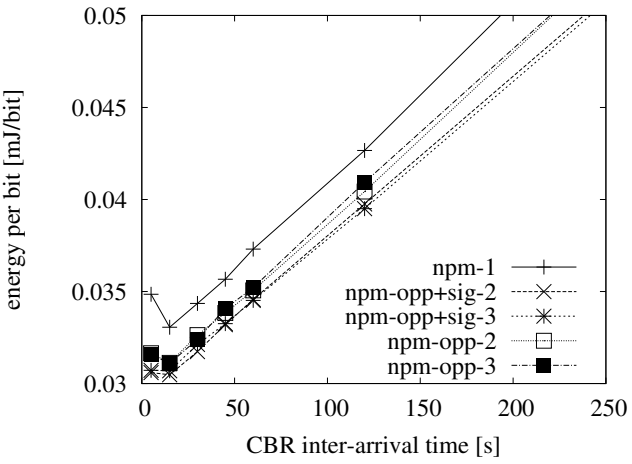

Figure 4.12: Energy for Any-MAC extended NPM: Realistic setup

anycasting for the wakeup signals of NPM using synchronization-based anycast extension in addition to having anycast control messages to gain both reduced wait times and increased opportunistic sending.

With `npm-opp`, we observed a 12% improvement in delay (see Figure 4.11) when k increases from 1 to 2. With more next hop options, `npm-opp-2` has higher probability of finding an awake next hop node than the original NPM without anycast (`npm-1`). Since the data sent opportunistically no longer needs to wait for a wakeup signal, `npm-opp-2` reduces the delay as it increases opportunistic sending in NPM. In addition to delay improvement, `npm-opp` also achieves similar improvement in energy (see Figure 4.12) since more opportunistic sending reduces the number of control packets to be exchanged between the nodes. Like X-MAC, the improvement from `npm-opp-2` to `npm-opp-3` is minimal.

With `npm-opp+sig`, we observed even higher delay improvements (see Figure 4.11), a 6% improvement over `npm-opp` and a 26% improvement over `npm`. In addition to improving the opportunistic sending with the `npm-opp` component, the `npm-sig` component enhances the wakeup signaling of NPM. With more next hop options, the `npm-sig` component has a higher probability of having recent information about at least one of its next hop options. Thus, `npm-opp+sig` reduces the number of times senders use full length preambles. By reducing the number of expensive full preambles, `npm-opp+sig` nodes observe relatively better energy improvements (see Figure 4.12) than `npm-opp`.

Thus, to summarize, by exploiting the inherent redundancy in WSNs, MAC protocols can effectively improve delay by enabling anycast. The energy improvement observed along with the delay improvements

indicates that Any-MAC extension incurs only minimal overhead. Our evaluations show that by using just one additional next hop node, Any-MAC extended protocols can gain promising improvements.

4.4 Conclusions and Future Directions

The contribution of this work is the design of Any-MAC [16, 15], a generic and low overhead anycast extension that exploits redundancy to improve delay of existing asynchronous MAC protocols. Any-MAC decouples the anycast component from the MAC protocol and allows any asynchronous MAC to enable anycast while still being appropriate for a particular network and application scenario. Our evaluations show that by leveraging the redundancy in dense networks and using just one additional next hop node, Any-MAC extended protocols can gain promising improvements in delay as well as energy.

Our current prototype works with shortest path routing protocol and hence uses only those nodes that require the fewest hops to reach the destination. Due to this restricted choice of the next hop nodes, the current Any-MAC prototype has limited opportunity to take advantage of the redundancy in the network. Any-MAC can be extended to take advantage of the suboptimal paths in the networks to enable further reduction in delay.

Chapter 5

Opportunistic AP Discovery in Dense Networks

In a dense WLAN, as more and more devices associate to an access point (AP) in the network, contention and collisions increase, with the predictable result of poor performance. An interesting and even more devastating problem arises even before the devices have associated with an AP. Our traces of many dense wireless networks revealed that over 40% of the traffic was a result of AP probes. Essentially, the simple act of scanning for available APs in dense networks brings the network to its knees. Instead of fighting with the density problem, we propose to improve performance by leveraging the available density. We have designed Dynamic Opportunistic Probing (Dynamo-Probing), an incrementally deployable solution that reduces the impact of AP probing through neighborhood collaboration in dense networks, but still provides timely discovery in sparse networks. The design of Dynamo-Probing is based on the fact that the scanning results of two nearby devices is likely to result in a high number of redundant discoveries. Dynamo-Probing exploits these similar fingerprints by introducing a novel AP scanning mode, Opportunistic Scanning (Opp-Scan). Essentially, by using Opp-Scan in a dense network, devices can discover most of their nearby APs simply by overhearing the responses from ongoing scans initiated by nearby neighbors without having to send any of their own probes. By monitoring on-going probing, Dynamo-Probing effectively manages the transitions between active and opportunistic scanning, resulting in low association delay in sparse networks and low probing overhead in dense networks.

5.1 AP Discovery

The poor quality of communication in dense networks is due to increased contention. The common perception is that if there were enough APs to support all wireless devices, contention and collisions would be significantly reduced and the problems would go away [9, 73, 53, 40, 71]. Although there are many solutions for load-balancing devices when multiple AP are available [42], simply adding more APs does not

solve the problem. The main limitation of these solutions stems from the fact that they only address which APs to associate with, not how to initially find them.

The problem in dense networks starts when devices decide to join a network and so actively search each channel with probe packets to obtain nearby AP information [55]. To understand the impact of scanning on dense networks, it is first necessary to delve into the actual scanning methods used by today's wireless devices. The goal of scanning is to discover the best AP in its wireless neighborhood. Without any other knowledge of network usage, "best" is typically defined as the nearby AP with the highest signal strength. Other metrics, like packet loss or delay, can also be used to decide when a device should start searching for a better AP. Most current devices start probing after the same packet has to be retransmitted three times, which indicates a "bad" channel. However and whenever a device decides to search for a new AP, the device must scan all wireless channels to gather current information about all nearby APs. Since the best scanning method depends on the local environment and the behavior of the device, various scanning modes have been proposed in the IEEE standards to support AP discovery in mobile environments. The effectiveness of these scanning modes can be evaluated by considering two metrics: the latency to acquire information about all nearby APs and the overhead incurred due to exchanging control packets to gather such information.

In this section, we discuss how the existing scanning solutions behave in dense networks. We start with the relevant details of the IEEE 802.11 AP scanning modes and improvements proposed in the literature. Finally, we discuss the actual scanning algorithms implemented in existing wireless device drivers.

5.1.1 IEEE 802.11 Scanning Modes

The IEEE 802.11 standard supports two scanning modes: passive scanning and active scanning [55]. Passive scanning is enabled by the periodic transmission of broadcast beacons by all active APs. Nearby wireless hosts can then passively obtain AP information by simply listening for these beacons. Although this approach has very low bandwidth overhead, a device must dwell in each channel long enough (100 - 200 ms) to ensure that it receives beacons from all existing APs on that channel. Since a device needs to scan all channels before selecting the best AP, the resultant discovery latency is typically too high for mobile device handoffs.

To reduce discovery latency, the IEEE standard proposes active scanning (see Figure 5.1). Devices accelerate the scanning process by actively probing each channel and collecting information from probe

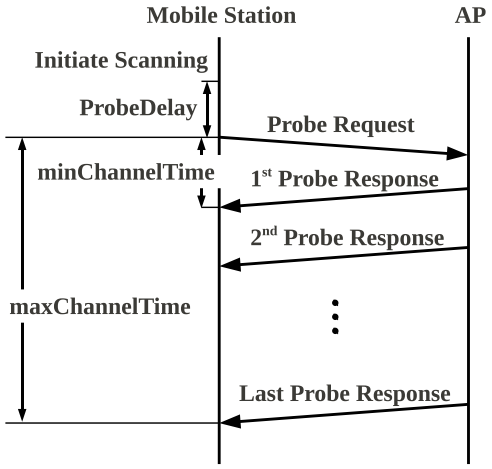

Figure 5.1: Active scanning process per channel

responses unicast back from nearby APs. To prevent probe collisions with ongoing transmissions, each device uses the underlying 802.11 mechanisms to scan the channel and avoid collisions. Essentially, each device listens for a DCF Interframe Space (DIFS), which is called `ProbeDelay` in the context of scanning, before sending its probe request. The actual time a device dwells in each channel is determined by two other parameters: `MinChanTime`, the time a device waits for an answer on an empty channel, and `MaxChanTime`, the time a device waits on a busy channel to gather responses from all APs. An effective solution must balance waiting long enough to collect all AP information on a busy channel while not waiting too long on unused channels. Since the IEEE standard does not specify any ideal value for these parameters, different device drivers use vendor specific values. However, it is recommended in practice to set `MinChanTime` to 4 - 7ms and `MaxChanTime` on the order of tens of milliseconds [55].

For an empty channel, active scan discovery latency is (`ProbeDelay` + `MinChanTime`) and for a busy channel, it is (`ProbeDelay` + `MaxChanTime`). Since the values of the active scan parameters are several orders of magnitude smaller than the maximum beacon interval, active scan is faster than passive scan and supports better handoffs. However, the low latency is achieved at the cost of increased overhead due to probe exchanges. While this overhead is not significant in sparse networks, our research reveals that these probes dominate traffic in dense networks and eventually result in network breakdown.

5.1.2 Improved Active Scan

Several approaches have been proposed in the literature to reduce network probing overhead and further improve discovery latency to enable seamless support for delay-sensitive traffic in mobile devices. However, none of these approaches directly address the problems associated with dense networks. By ignoring the problem of probing and contention, some of these approaches even end up increasing the probing overhead beyond that of the basic active scanning, resulting in increased contention.

To eliminate the probe floods expected in highly mobile networks, devices can collaborate and acquire AP information by enabling broadcast probe responses [54, 60] instead of unicast responses. Although proposed as an extension to 802.11, there are three functional reasons why broadcast probe responses were not approved by IEEE [60]. First and foremost, broadcasting the probe responses results in unreliable AP discovery since 802.11 broadcast messages are not sent reliably. Second, the proposed approach requires changes to the APs, which is highly discouraged by the IEEE since it is not easy to deploy. Finally, all devices, including the legacy devices, receive all broadcast probe responses, resulting in increased energy consumption even when they are not scanning. While this energy consumption is not prohibitive, there are always concerns about solutions that increase energy consumption. In the end, the negatives far outweighed the positives of broadcast probe responses.

Improved association delay in dense networks would be welcome. However, it is difficult to achieve orthogonally to solving the problem of probing and contention. In sparse networks, the key to reducing delay for active scanning is to limit the number of scanned channels and to select appropriate values for the scan parameters. Devices can adopt *selective scanning* to limit the number of channels to be scanned (i.e., QuickWiFi [31], channel masking and AP caching [58], and Neighbor Graphs and Neighbor Pruning [69, 59, 56, 57, 70]). Using channel masks, a device masks off the empty channels from the list of channels to be scanned. Similarly, QuickWiFi only scans orthogonal channels, reducing association delay. Although effective in sparse networks, both approaches fail to reduce scanning latency in dense networks since there are likely APs on all available channels. Devices can further limit discovery latency by caching adjacent AP addresses and probing only the best adjacent APs on selected channels. Since none of the APs in a dense network is likely to provide better performance, the frequency of probing remains the same. Finally, Neighbor Graph (NG) and NG-pruning approaches face the same problems as AP caching in dense networks. Moreover, these solutions incur additional overhead at each device for maintaining the NG tables.

Determining a tighter bound for `maxChanTime` is important to ensure improved scan latency. Traditional active scan incurs long latency in AP-dense networks since it waits on a channel until it receives probe responses from all APs on that channel. D-Scan [80] proposes shorter values (less than 30 `msec`) for `maxChanTime` by claiming that APs with higher signal strength typically respond to probe requests early. While this is may be true in sparse networks, replies in dense networks get arbitrarily delayed due to background traffic. Hence, a short `maxChanTime` in dense networks will only result in unsuccessful scanning, missing most of the potential probe replies. Our experiments show that even with the regular values for channel time parameters, most probing in dense networks results in unsuccessful scans where no APs are found.

Finally, devices can avoid incurring scanning latency at all by pre-scanning all channels [22, 85, 23, 66] and constantly maintaining updated information about all nearby APs. However, pre-scanning requires the devices to scan more frequently than active scan, which incurs higher probing overhead and stresses the dense network even more.

5.1.3 Scanning Modes in Practice

Existing wireless device drivers implement a combination of active and passive scanning, which we will call Hybrid-Scan in the rest of this paper. In Hybrid-Scan, a device starts in passive mode and switches to active mode as soon as its MAC protocol detects poor channel conditions, typically indicated by three unsuccessful transmissions of a packet. While scanning, devices cache the service set identifier (SSID)s of their preferred networks. Unlike the IEEE standard for scanning, devices scan each channel in two rounds. In the first round, devices probe each channels to find APs in their preferred networks by embedding the SSID of one preferred network inside a probe request and repeating this for all preferred networks. Only APs in the preferred network respond to such targeted probes. This targeted probing continues until the device finds a better AP in one of its preferred networks, or it has exhausted the entire list of known networks, which may be quite long. If no good AP is found on a known network, devices start their second stage of probing and send a broadcast probe request containing NULL SSIDs, which all APs reply to. This two-stage active scanning is similar to AP caching, and so faces similar limitations. Essentially, devices end up sending many targeted probes for networks that are probably not even nearby, increasing the total probing overhead, and when they do not get any responses, end up probing for all APs, increasing overhead and delay even

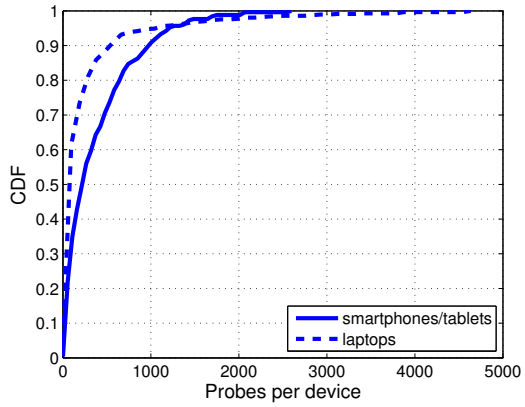

Figure 5.2: CDF of probes sent per device

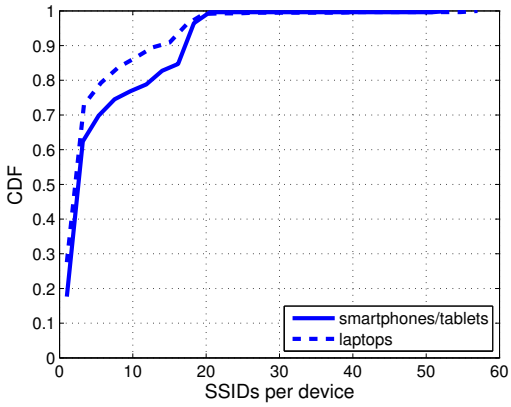

Figure 5.3: CDF of SSIDs searched per device

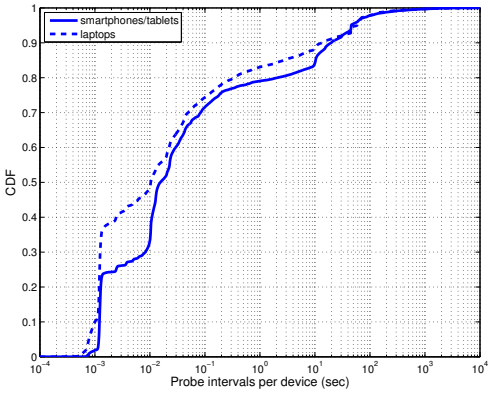

Figure 5.4: CDF of probe intervals per device

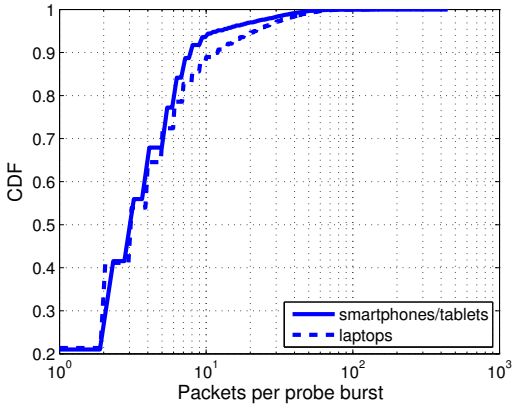

Figure 5.5: CDF of probe burst length

further.

In the end, the channel conditions in dense networks do not improve even after a successful scan and the existing scanning results in cycles of probe storms, eventually introducing more contention into the network. Our analysis of the traces we collected in real dense networks (described in the next section) reveals the excessive probing problem and reinforces the need for a solution to this practical problem.

5.2 Traces of Dense Networks

Although it is well-known that 802.11 does not work well in dense networks, it is difficult to reveal the real reasons without looking at how devices interact with APs. We collected wireless traces during `IEEE INFOCOM 2012`, at two major airports with open Wi-Fi, and in large university classrooms with more

Smartphones and tablets	255
Laptops	569
Total devices	824

Table 5.1: Distribution of devices in the INFOCOM Keynote Traces

than 200 students using the open source packet analyzer tool WireShark [26] on a Lenovo ThinkPad T410 laptop. For the INFOCOM traces, the network was very dense during the conference keynote session. We also observed high network activity in the airport traces collected close to terminal gates right after flight arrivals and before flight departures. The classroom setup had sufficient density to trigger similar probing problems. Given the similarity of the traces, we only present results from the analysis for the INFOCOM 2012 keynote traces in this section.

The two hour long opening and keynote sessions of IEEE INFOCOM 2012 were held in a very large room that was fully packed with attendees. The room was covered by three spread-out APs all operating on the same channel with SSID=Infocom. Although this may not have been the ideal set-up, it is what was provided by the venue. With so many devices in one space, even if the APs had been on different channels, they still would have been overloaded and all of the devices would still have been scanning all of the channels.

Our traces revealed that most devices initiated connection within the first five minutes of the session and stayed until the end of the two hour long session. When we analyzed the traces, we identified 824 unique devices within the coverage range of our monitoring laptop representing multiple types of wireless devices (laptop, tablet, smartphone, etc.). By analyzing the unique MAC addresses, we were able to identify the distribution of laptops vs. smartphones and tablets (see Table 5.2). This distinction is important since the drivers in laptops use different scanning parameters than phones and tablets. Since smartphones and tablets are expected to be more mobile than laptops, the parameters are set to scan the channels more frequently to support seamless roaming.

During the keynote session, network discovery and connection were both very slow, and all devices experienced very poor performance. The ineffective network discovery process was highlighted in our traces, which revealed that the majority of the packet transmissions were probe packets. Despite an expectation of fewer probes from laptops, our traces show an unexpectedly high number of probes for all types of devices. On average, each smartphone/tablet transmitted 373 probe requests during the session, whereas each laptop

transmitted 227 probes (see Figure 5.2). Although excessive probing is higher in smartphones and tablets than in laptops, probe packets from all devices dominate traffic in all of our traces.

During the keynote, the only available network was `SSID=Infocom`. Therefore, all devices eventually connect to the same SSID, wasting network bandwidth that was used to probe for nonexistent networks. This behavior can be seen by looking at the the number of unique SSIDs probed by each device (see Figure 5.3). Since smartphones and tablets cross more networks than laptops, they typically have longer lists of known networks than laptops and send more probes looking for APs in each round. Although, on average, the smartphones and tablets probed for 5 SSIDs and the laptops probed for 4 SSIDs, 20% of the smartphones and tablets probed for more than 10 different SSIDs, while only 13% of the laptops probed for such high number of SSIDs.

Since all of the devices continually scanned, we can infer that despite finding and settling on an AP, none of the devices were satisfied with their connectivity. The result was massive bursts of probes. Additionally, many of the probes were sent back-to-back from a single device, with more than 50% of the probes from the same device less than 10 ms apart (see Figure 5.4). These bursts are formed by the one-per-preferred-network back-to-back targeted probes at the start of each scan round. The traces also reveal that 20% of the devices start their next scan within 1 `second`. Figure 5.5 shows that 90% of the probe bursts were 10 packets or less, while only 20% were isolated probes.

Bringing all of this together, the cycles of probe storms is the main reason for the continuous deterioration of channel conditions in dense networks. Such probe storms cannot be handled by improving active scan (discussed in Section 5.1.2). In fact, we will next show that the solution to this problem is in leveraging density instead of ignoring it.

5.3 Dynamo-Probing

In a dense network, the best AP for neighboring devices will likely be the same, indicating opportunities for collaboration in acquiring AP information. Such collaboration can reduce contention in the network by reducing probe storms and ultimately improving communication quality. With this in mind and in response to the limitations of current scanning solutions, we have designed Dynamic Opportunistic Probing (Dynamo-Probing), which is designed to leverage the active scans and similar AP fingerprints of nearby devices to complete its scan process in dense networks, significantly reducing probing overhead and contention. How-

ever, since collaborative opportunities are not available in sparse networks, Dynamo-Probing is designed to adapt to the density of the network, falling back on per-device active scans in sparse networks.

In this section, we describe how Dynamo-Probing uses our novel opportunistic scanning mode, Opp-Scan, and Dynamo-Probing's strategy to handle dynamic changes in density.

5.3.1 Scanning using Dynamo-Probing

To be effective in both dense and sparse networks, Dynamo-Probing dynamically switches between two scanning modes: active scan and our novel Opportunistic Scan (Opp-Scan). We designed Opp-Scan to take advantage of any high network density and enable low overhead and low latency scanning. In contrast, using active scan enables the devices to quickly obtain AP information in sparse networks. By dynamically switching between the two modes, Dynamo-Probing eliminates the high probing overhead from active scan in dense networks and the high discovery latency from Opp-Scan in sparse networks.

Similar to traditional scanning, at the beginning of a scanning process, a device using Dynamo-Probing wait for `ProbeDelay` before broadcasting a probe request packet (see Figure 5.6). While waiting for `ProbeDelay` to expire, if a device using Dynamo-Probing receives a broadcast probe request from another device, it immediately enters Opp-Scan mode and cancels its transmission of its broadcast probe request. If no broadcast probe is heard, the device initiates an active scan, broadcasting its probe request. In both modes, devices obtain AP information from beacons and by overhearing the results of nearby devices' active scans whenever possible. Overhearing the data packets during scanning does not help to eliminate probing since these packets only contain the address of the AP and do not contain the additional information required to associate with an AP such as the supported rates. Since devices only use overhearing when they are searching for a new AP, they do not have any overhearing cost during their normal data transmissions. This minimal overhead is worth the significant improvement in terms of network performance.

After entering Opp-Scan mode, the device listens for a `maximum channel time` before ending its scan on a busy channel. During this time period, the Dynamo-Probing device extracts AP information from all overheard probe responses directed towards any nearby device. After the timeout, the device turns off overhearing and moves on to the next channel to scan. The decision whether to perform Opp-Scan or active scan must be taken on a per channel basis. Therefore, the device repeats this process on every channel. Once the device collects AP information on all channels, it selects its best AP and stops scanning.

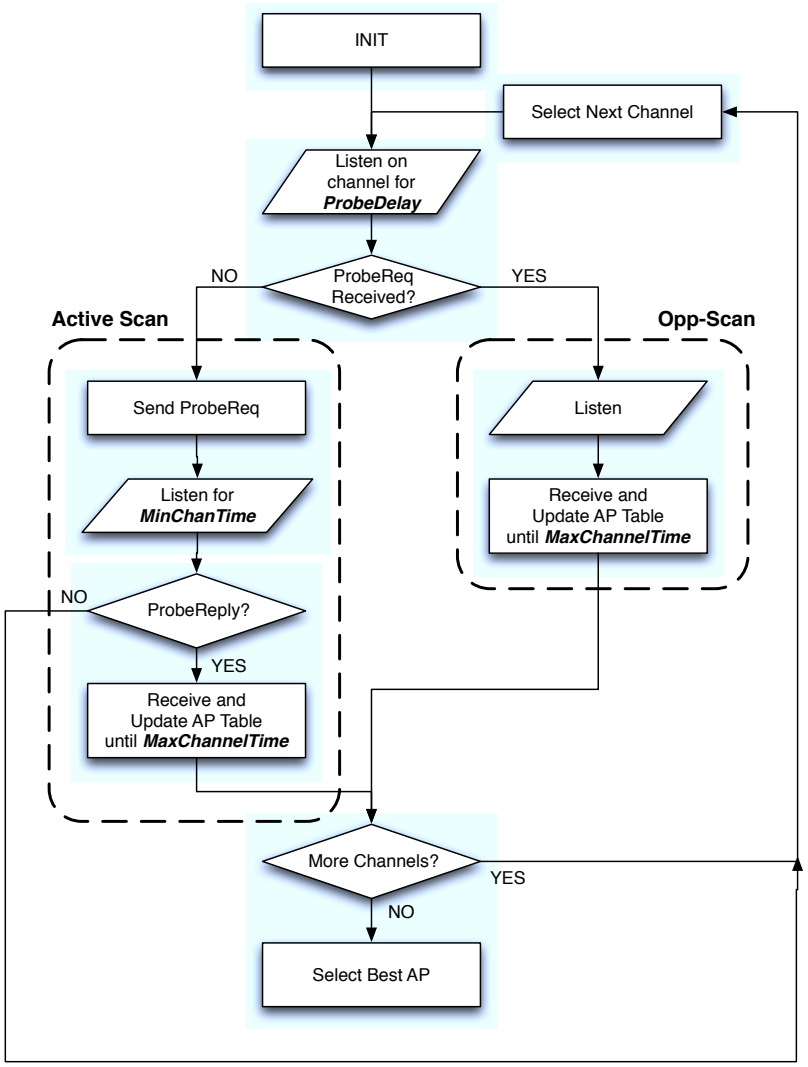

Figure 5.6: Dynamo-Probing

Although Dynamo-Probing is conceptually similar to the broadcast probe response proposal [54, 60] discussed in Section 6.1, it has two major benefits. First, Dynamo-Probing enables the AP to continue using unicast probe responses, maintaining high reliability for such responses. Second, Dynamo-Probing is entirely a client-side solution and does not influence the operation of other devices in the network. An individual device in a network of active scanning devices could use Opp-Scan to obtain most of its AP information without sending any of its own probes, enabling improved network performance even if only a subset of the devices use Dynamo-Probing. Finally, by only turning on overhearing when the device is intentionally scanning, Dynamo-Probing avoids incurring unnecessary energy overhead processing broadcast probe responses during non-scanning periods. These differences make Dynamo-Probing a more practical

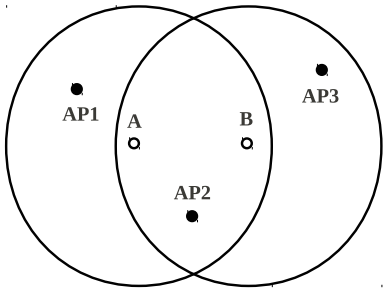

Figure 5.7: Diverse fingerprint of wireless network

and more effective solution than enabling broadcast probe responses.

5.3.2 Dynamo-Probing in Dynamic Networks

Although Dynamo-Probing focuses on using Opp-Scan in dense networks, it must be able to quickly use active scan in two situations: first, in sparse networks when there is no ongoing active scans, and second, to obtain a full AP list since some close devices' AP fingerprints may not completely overlap (see Figure 5.7). The challenge is to determine when to use Opp-Scan and when to fall back to active scan.

In sparse networks, the decision is simple—use active scan when the device does not detect any ongoing active scans. However, the decision to switch modes is not trivial in dense networks. Wireless devices have large transmission ranges. Thus, while neighboring devices may share many APs, there may also be AP outliers that are only in range of one or a few of the devices. In Figure 5.7, when A uses Opp-Scan and achieves AP information from B's active scan, it can only identify AP2. Similarly, B fails to obtain information about AP3 when using Opp-Scan. To overcome this limitation of Opp-Scan, Dynamo-Probing must alternate between the two modes in dense networks.

Since a device always waits `ProbeDelay` before sending a broadcast probe and resolves the contention with other devices using 802.11 backoff, the behavior of the devices can be controlled by setting `ProbeDelay` at each device appropriately. If all devices have the same value for `ProbeDelay`, 802.11 contention management insures that all devices eventually win the contention and send their active probes, resulting in a somewhat random distribution of scanning. However, with fixed values of `ProbeDelay`, short-term fairness is not guaranteed and some devices may not get a chance to scan and so may miss out on some AP opportunities in their neighborhood.

An alternative is to set `ProbeDelay` at each device to provide for some priority. Since Dynamo-

Probing is designed to be collaborative, it should ensure that each device in a neighborhood performs relatively equal numbers of active scans. Dynamo-Probing prioritizes the devices based on their scan history. Devices can have two possible `ProbeDelay` values: the base value x and $2x$. On a new channel, a devices uses the base `ProbeDelay`, ensuring quick discovery in sparse networks. However, each time a device scans in active mode, it sets `ProbeDelay` to $2x$ for its next scan. The longer wait time increases the device's probability to use Opp-Scan instead of active scan on its next scan. The device resumes its base `ProbeDelay` value after scanning in Opp-Scan mode. Although only using x and $2x$ for `ProbeDelay` may not seem to provide enough distinction, the use of 802.11 contention resolution introduces some randomness as to which device ends up sending an active probe. Therefore, it is enough to distinguish between devices that just did an active scan and ones that just used Opp-Scan. Additionally, extending `ProbeDelay` beyond $2x$ can negatively impact discovery delay. In our experiments, we evaluated both the static and the priority-based approaches to identify the best solution.

5.4 Evaluation

The goal of our evaluation is: (1) to demonstrate the **effectiveness** of Dynamo-Probing in obtaining AP information by collaboration, (2) to evaluate its **efficiency** in improving communication quality in dense networks by eliminating unnecessary probe storms, (3) to investigate its ability to seamlessly **adapt** to varying network density, and finally (4) to test its **interoperability** with devices using legacy scan modes. Success in all four domains establishes Dynamo-Probing as a practical and effective solution to make the dense networks work.

For our evaluations, we implemented Dynamo-Probing in `ns-2`. Our simulation environment was set up such that it closely approximated our collected traces. The only major difference between the real traces and our simulations is that `ns-2` only simulates communication on a single channel; hence, our simulation results captured the effectiveness of Dynamo-Probing on a per channel basis. However, this limitation of `ns-2` neither limits nor favors the operation of Dynamo-Probing. We expect Dynamo-Probing to achieve an increased aggregate benefit when the devices repeat their same scanning mechanisms on multiple channels.

We compared the performance of Dynamo-Probing to Hybrid-Scan, which is implemented in most wireless drivers (described in Section 5.1.3). For both protocols, retransmission of the same packet 3 times indicates a poor channel and triggers AP scanning. In hybrid mode, a device starts in passive mode and turns

Parameters	Values
`ProbeDelay`	0.1 msec
`maxChanTime`	11 msec
`minChanTime`	5 msec
`Passive Scan chanTime`	120 msec

Table 5.2: Scan parameter values

on active scanning after detecting a bad channel. In contrast, Dynamo-Probing starts in active mode and switches to either active scan or Opp-Scan depending on the opportunity to exploit results from other ongoing scans. Table 5.4 lists values of the other scan parameters for both Hybrid-Scan and Dynamo-Probing. We have simulated Dynamo-Probing with longer values for its scan paramaters, but have obtained similar results. In our simulations, devices always send broadcast probe requests instead of sending both targeted and broadcast probes during each scan as typically implemented in all devices. We expect to experience even greater improvements in this two stage scanning compared to our simulations since Dynamo-Probing will have more opportunities to eliminate both the unnecessary broadcast and targeted scans. We ran Dynamo-Probing with the static and priority-based `ProbeDelay` to evaluate the effectiveness of both mechanisms and to identify the best approach. Since the cycle of probe stroms is one of the main reasons for poor performance in dense networks, we investigate whether Dynamo-Probing would benefit from eliminating probing alltogether by switching to passive mode by comparing its performance to the passive scan mode.

5.4.1 Performance Metrics

When Dynamo-Probing is effective in obtaining AP information by collaboration, it eliminates unnecessary probes during its scanning process. We measure Dynamo-Probing's effectiveness by analyzing the *total probes*, which includes both probe requests sent by the devices and the probe replies sent by the APs. By comparing the number of probe requests sent to the number of probe replies received, we estimated probability of success for each scan. This is also an indirect assessment of the quality of AP information obtained from each scan.

The efficiency of scanning is captured by the *average association time* for both approaches. This metric represents the time a device requires on average to complete its scanning, authentication and association process. Since the scanning latency may vary from device to device depending on its traffic and its position in the network, we also monitor the *minimum association time* and the *maximum association time*. When

a device is not successful in discovering any AP, either due to delayed probe responses or due to non-overlapping AP fingerprints between nearby devices, it must wait until a later successful scan to complete its scan. The *average association time* indirectly captures this delay overhead.

Finally, the ultimate goal of Dynamo-Probing is to reduce contention due to recurring scanning and allow the dense networks to actually support some communication. The success of Dynamo-Probing in making the network actually work can be captured by monitoring *packet drop ratio* and *throughput*.

5.4.2 Network and Traffic

We setup our simulation networks within a 50 m $\times$ 50 m confined space to match the dense networks observed in the `INFOCOM 2012` traces. In addition to validating Dynamo-Probing's effectiveness in dense networks, it is also important to make sure that the devices perform well in sparse networks. Hence, we varied the number of devices in our simulations starting from 20 devices up to a density of 250 devices. While the network with 20 nodes was a representative of the lightly-loaded workshop sessions at `INFOCOM`, the 250-node network represented the busy opening and keynote sessions at the conference.

The value of *total probes* depends on the number of devices sending probes and the number of APs on the channel. In a "well managed" network, most devices are within the transmission range of only one AP. Hence, a broadcast probe request from the device results in one unicast probe response from its AP. However, for devices that are within the range of multiple APs, a broadcast probe request causes all APs in range to reply. To simulate this in our networks and to assess Dynamo-Probing's performance in a network where nearby devices may have non-overlapping AP fingerprints, we varied the number of APs in the network from 1 to 4. In the simulations with multiple APs, the APs were placed and the transmission ranges of the devices were setup appropriately to avoid AP to AP collisions. Since the results with the different AP setup had similar trends, we present the results obtained with 4 APs.

Traffic usage largely varies from user to user depending on the number and types of applications they are running on their devices. While applications such as email, web browsing, and DNS generate small transfers, multimedia streaming and maps typically generate large transfers. To simulate the patterns of web traffic in real networks, each device in our simulations generated traffic according to distributions obtained from Internet traffic measurements [32, 34, 17]. We ran separate simulations for two different datasets (see Figure 5.8). `Dataset1` represents traffic in lightly loaded networks and consists of a large number of

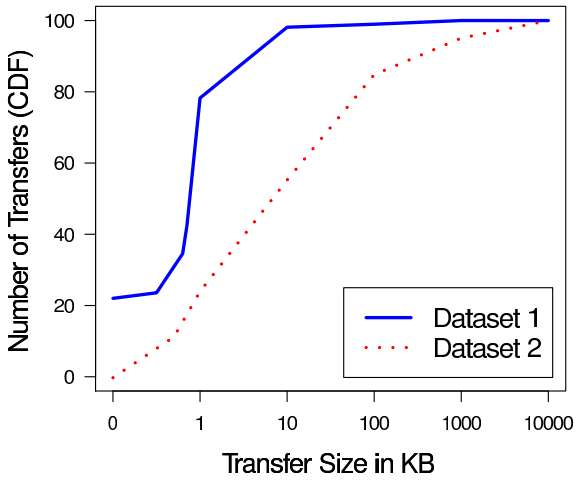

Figure 5.8: Network traffic distribution

small transfers. `Dataset2` represents relatively heavy traffic consisting of a mix of both small and large transfers. The reason for not using the traffic pattern extracted from the `INFOCOM` traces in our simulations is that the traffic was polluted with recurring probes and was not an ideal representation of the actual web access pattern.

Each simulation was run for a 30 min of simulated time. Devices joined the network at a random time within the first 60 sec and generated traffic for 25 min. Results presented in this section are obtained by averaging the results from multiple simulation runs.

5.4.3 Effectiveness

Effectiveness is evaluated using two metrics: *total probes* to capture the total probing overhead and *association time* to capture its discovery latency. An effective solution should incur low overhead for scanning and quickly discover nearby APs.

Dynamo-Probing eliminates unnecessary probes by entering Opp-Scan mode and extracting AP information from ongoing active scans of nearby devices whenever possible. In dense networks, where there are many devices, there will be plenty of such opportunities. When compared to Hybrid-Scan, Dynamo-Probing starts to show significant improvements in *probing overhead* when the network has more than 100 devices (see Figure 5.9). Dynamo-Probing reduces the total probing overhead by 36% when the network has 250 devices. The improvements are similar for both static and priority-based approaches for Dynamo-Probing. In addition to eliminating the unnecessary probe exchanges, Dynamo-Probing also improves the quality of

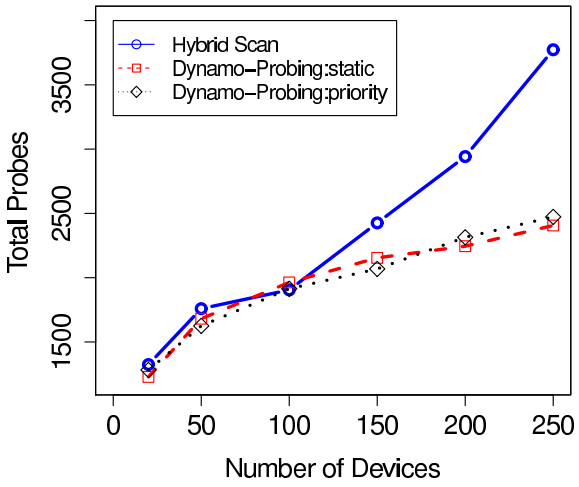

Figure 5.9: Probe overhead

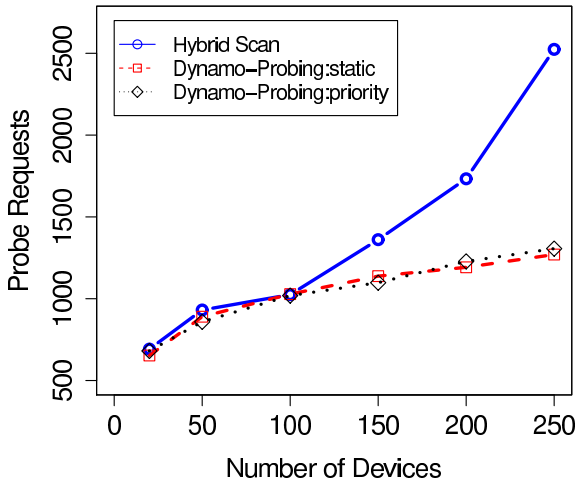

Figure 5.10: Probe request

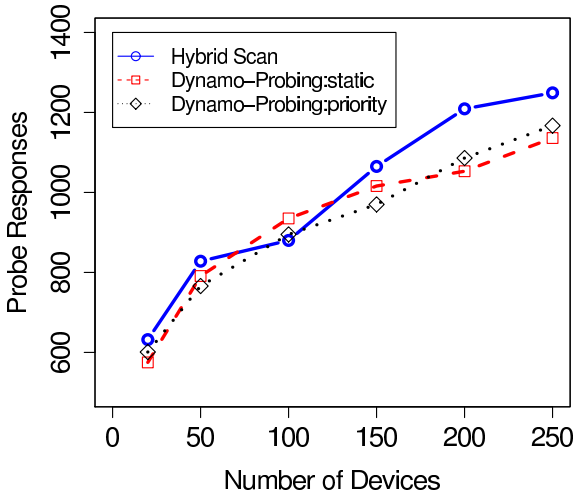

Figure 5.11: Probe responses

each scan. While with Hybrid-Scan, only 50% of the total scans resulted in successful AP discovery, about 90% of the scans were successful in the 250 node network with Dynamo-Probing (see Figure 5.10 and Figure 5.11).

One of the main reasons for an unsuccessful scan is probe request collisions, which are very common for Hybrid-Scan in dense networks since all devices experience packet drops and multiple devices start their scans at the same time. However, with Dynamo-Probing, only one device in the neighborhood starts to scan in active mode, while the rest start Opp-Scan. This definitely reduces the probability of probe collisions. Since a device keeps on scanning until it finds a better AP, the higher success rate of Dynamo-Probing's scan also reduces the number of future scans. Hence, Dynamo-Probing not only eliminates current unnecessary

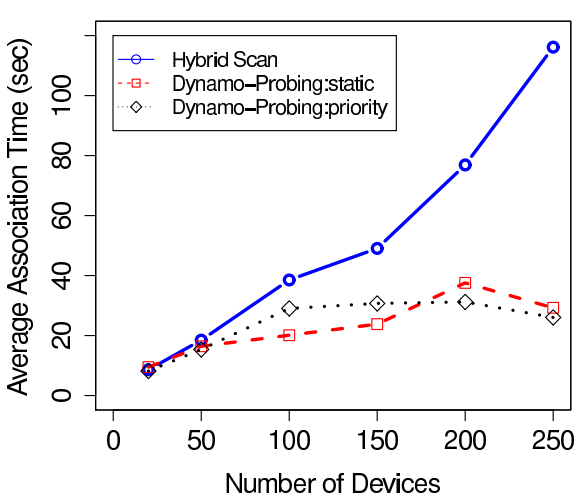

Figure 5.12: Average association time

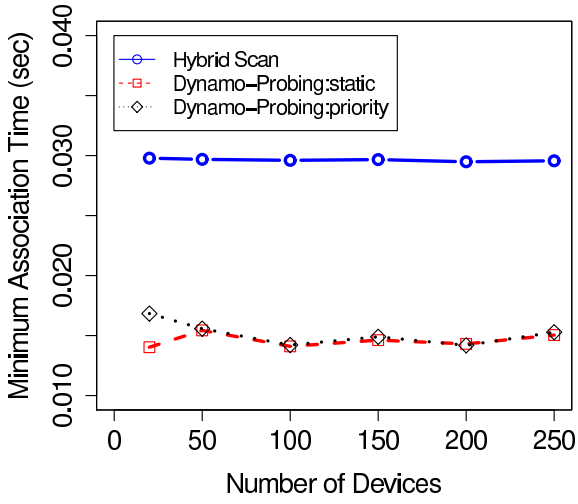

Figure 5.13: Minimum association time

probes, it also reduces the need for additional future scans.

Dynamo-Probing performs surprisingly better in terms of discovery latency when compared to Hybrid-Scan in dense networks. Although a successful scan by Dynamo-Probing is expected to take as long as an active scan, the higher success ratio of Dynamo-Probing eventually results in very low *average association time* compared to Hybrid-Scan. In contrast, Hybrid-Scan requires multiple scan rounds for a successful AP discovery and results in high latency. To identify the reason for this large percentage of unsuccessful scans by Hybrid-Scan, we analyzed the time between a probe request and its reply. Our analysis showed that the scan latency vaired between 4 msec to 140 msec. In dense networks, the increased contention due to probe storms often pushed the probe replies after the 11 msec `maxChanTime` for Hybrid-Scan which resulted in the increased unsuccessful scans. Dynamo-Probing devices can improve the average association time by using higher values for `maxChanTime`. Our simulations showed that even with an 11 msec `maxChanTime`, Dynamo-Probing starts experiencing the better latency in sparse networks when the network has only 20 devices by eliminating contention from concurrent probe storms (see Figure 5.12). At 250 nodes, Dynamo-Probing improves the latency by 78%. While the improvements by both approaches are high, the priority-based approach achieves better latency (78%) than the static approach (75%). By making sure all devices perform equal number of active scans, the priority-based approach enables faster AP discovery than the static approach for those devices that have neighbors with non-overlapping AP fingerprints.

Dynamo-Probing even offers a lower *minimum association time* and a lower *maximum association time* than Hybrid-Scan (see Figures 5.13). The *minimum association time* of Hybrid-Scan depends on the time it

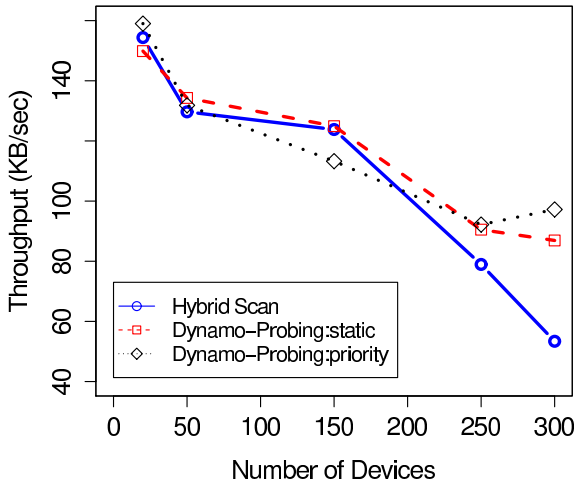

Figure 5.14: Throughput

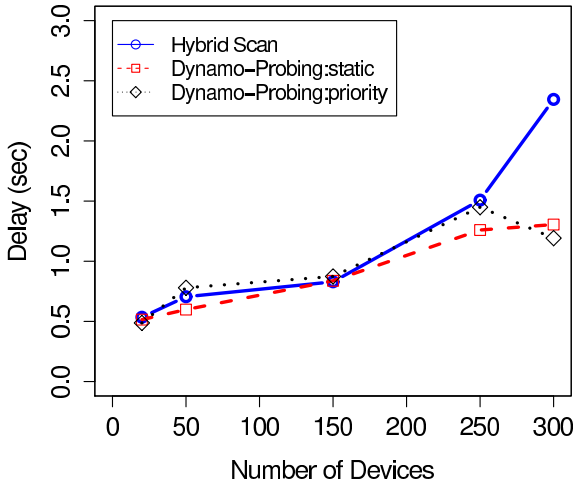

Figure 5.15: Delay

takes to scan in passive mode and in active mode. In dense networks, these devices experience unsuccessful scans and arbitrary delay due to probe storms pushing the time for a successful active scan beyond one beacon interval, resulting in a *minimum association time* of 300 msec. In contrast, by controlling the probe storms and better managing contention, Dynamo-Probing can associate with an AP as fast as 150 msec. Similarly, the higher success of Dynamo-Probing also results in an at least 15% lower *maximum association time* than Hybrid-Scan.

Besides demonstrating its effectiveness in dense networks, Dynamo-Probing is also effective in sparse networks where the scans are almost always successful (see the lower number of devices in Figures 5.12 and 5.13. In sparse networks, contention is low, fewer packets get dropped, and there are very few opportunities for using Opp-Scan. Hence, the improvements observed by Dynamo-Probing in such networks are also comparatively low. However, even in a network with only 20 devices, Dynamo-probing reduces the *probing overhead* by 7% and the *average association time* by 5%.

5.4.4 Efficiency

By managing contention, Dynamo-Probing enables improvement in the quality of communication by improving throughput and reducing delay. By eliminating the unnecessary probe storms in dense networks, Dynamo-Probing reduces contention and achieves maximum improvements. In contrast, Hybrid-Scan starts overloading the network with probe packets at a density of 150 devices and results in a steep decrease in throughput (see Figure 6.1) and a steep rise in delay (see Figure 6.2). When compared to Hybrid-Scan at the

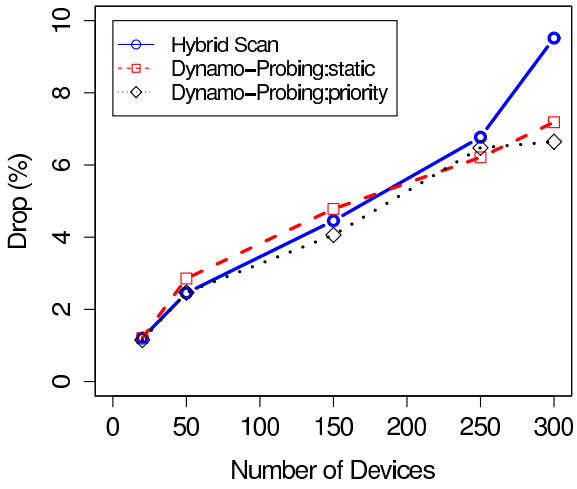

Figure 5.16: Drop ratio

density of 300 devices, Dynamo-Probing devices experience 82% higher throughput and 50% lower delay. The improvements are greater for the priority-based approach than the static approach since the devices in the priority-based approach have better quality AP information and hence use fewer probes compared to the static approach. Even in sparse networks with 20 devices, Dynamo-Probing achieves a 3% improvement in throughput and a 5% reduction in delay.

The efficiency of Dynamo-Probing is not limited to improved throughput and delay, it also reduces packet drops. Since we ran our simulations with TCP flows, the packet drops experienced by the devices were not high (see y-axis of Figure 6.3). However, Dynamo-Probing still managed to reduce the *packet drop ratio* by 30% when the network had 300 devices. We expect to observe even greater improvements in packet drops when the devices run UDP flows.

5.4.5 Adaptability

The detailed analysis of Dynamo-Probing's performance in the different network densities show that Dynamo-Probing is effective and efficient in all types of networks. While Dynamo-Probing thrives in dense networks by achieving immense improvements in all performance metrics, it also demonstrates some improvements in sparse networks. Hence, the results support our claim that devices implementing Dynamo-Probing is adaptive to varying density.

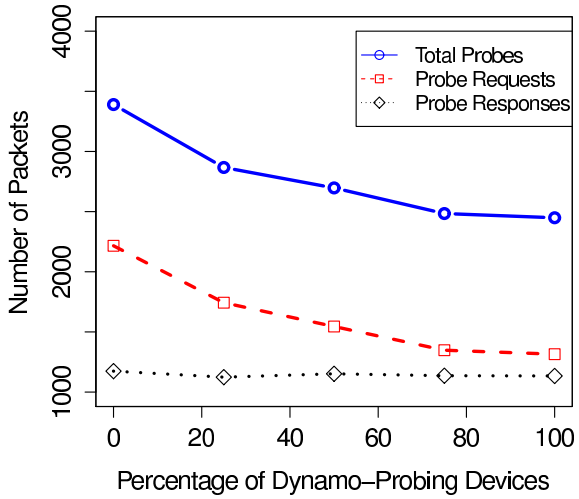

Figure 5.17: Total probes in mixed network (density=250 devices)

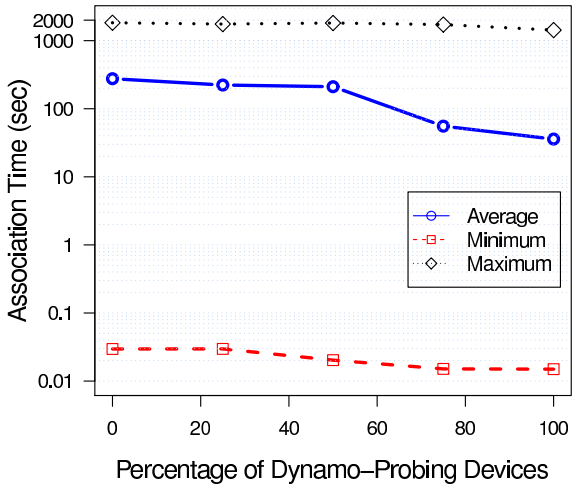

Figure 5.18: Association time in mixed network (density=250 devices)

5.4.6 Interoperability

One of the biggest strengths of Dynamo-Probing is that it can be deployed incrementally and, although performance will improve at higher adoption rates, lower rates still enable significant improvements. To demonstrate Dynamo-Probing's effectiveness and efficiency in a network that contains a mix of Hybrid-Scan and Dynamo-Probing devices, we ran simulations for different network densities while varying the percentage (0%-100%) of Dynamo-Probing devices in the network. 0% indicates all devices are running Hybrid-Scan and 100% indicates all devices are running Dynamo-Probing.

To achieve benefits from Dynamo-Probing, it is not essential to deploy Dynamo-Probing in all devices in the network. Even a few Dynamo-Probing devices can result in more effective scans in dense networks. By deploying Dynamo-Probing in only 25% of the devices in a network with 250 devices, the number of active scans can be reduced by 21% (see Figure 5.17). The improvement gradually increases as more devices deploy Dynamo-Probing, eliminating 40% of the scans when all devices implement Dynamo-Probing. It is quite interesting that Dynamo-Probing can reach within 1% of this maximum *total probe* improvement for this network with a 75% deployment.

The gradual increase in the success ratio of scans also results in significant improvements in *association time*, reducing the *average association time* by 19% with only a 25% Dynamo-Probing deployment (see Figure 5.18). The biggest improvements in *average association time* is observed when more than 50% of the devices implement Dynamo-Probing. The same trend is also observed in throughput performances for both

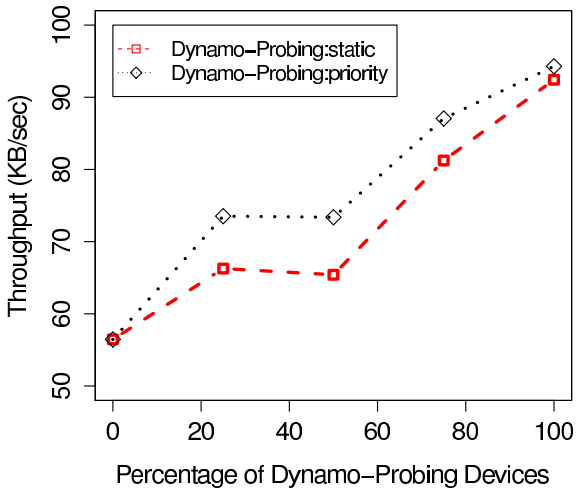

Figure 5.19: Throughput in mixed network (density=250 devices)

static and priority-based Dynamo-Probing (see Figure 5.19). As expected, the improvements for the priority-based approach is always higher compared to the static approach (31% and 18% at 25% deployment) in dense networks.

In sparse networks where probing does not incur too much overhead or overwhelm network communication, the maximum improvements are achieved at 25% deployment and only slowly improves as more Dynamo-Probing devices are deployed. The same trends are observed for all performance metrics (total probes, association time, throughput, delay and drop ratio).

5.4.7 Dynamo-Probing vs. Passive Scan

Passive scan eliminates probing overhead. However, its long discovery latency causes a more devastating problem. Devices experience extended periods of inactivity when they scan. During the scan process, data transmissions cannot progress since the device is not associated with any AP on the channel being scanned. The data transmission stalls until the device has found a better AP and associates to that AP, or the device concurs that its current AP is the best available one. During passive scan, a device must dwell in a channel for at least a beacon interval time period, whereas Dynamo-Probing devices scan on a channel for a `maxChanTime` period. Our simulations show that with 3 channels to scan, passive scan ends up incurring 15 times more offtime than Dynamo-Probing (see Figure 5.20). The ratio will be even higher if the device scans more channels. Interestingly, this long period of inactivity not only affects the devices in dense networks, the impact is significant even in sparse networks. Each device incur 11 times higher

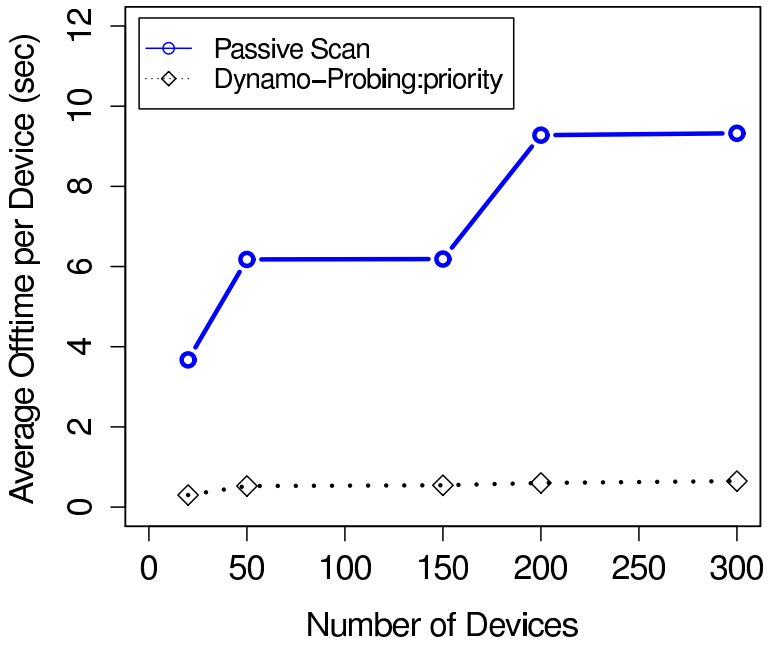

Figure 5.20: Overhead of Passive Scan

inactive time on an average when compared to Dynamo-Probing in sparse networks. Hence, passive scan cannot support seamless handoff and is not appropriate for mobile devices.

5.5 Conclusions and Future Directions

The contribution of this work is twofold. First, we identified the main reason for the network breakdown at high density. The traces collected from dense networks revealed that the mobile devices in dense networks are constantly searching for better APs to associate with even though they may not be roaming. Unfortunately, simply finding a new AP does not solve the problem. Since all nearby APs are probably overloaded and so do not provide any better channel conditions, devices experience no improvement even after successful handoffs, probing does not stop and the devices continually search for better APs. Finally, since all devices are experiencing poor communication quality, they all start scanning at almost the same time. The aggregate effect of this is a flood of AP probes causing even more contention in the network. Eventually, all devices enter a vicious cycle of probing, wasting the limited available bandwidth and causing the network to become almost inoperable. Second, we propose a solution to this problem by leveraging high density. Our solution, Dynamo-Probing, opportunistically obtains AP information from nearby active scans. By eliminating unnecessary scans in all types of networks, Dynamo-Probing not only reduces the probing overhead,

but also manages contention and prevents the network from thrashing at high density. Even with incremental deployment, Dynamo-Probing can provide significant improvements in scanning and providing better communication quality in dense networks. All these characteristics make Dynamo-Probing an effective, efficient and readily deployable scan solution.

The next step for Dynamo-Probing would be to implement it in real devices and evaluate its effectiveness and efficiency in a dense network testbed. The scanning algorithm itself can be extended in a number of ways. The current design of Dynamo-Probing only focuses on eliminating the probing overhead due to concurrent broadcast probes. We expect that devices would benefit even more if Dynamo-Probing were extended to eliminate the concurrent directed probing targeted towards their preferred networks. The main problem with scanning in dense networks is that devices keep on probing even though they keep finding the same APs. Hence, extensive research is required to identify what should be the probing ideal—whether Dynamo-Probing can eliminate all unnecessary scans in dense networks especially when the devices know that they are not moving and the channel is not changing. One possible approach to eliminate the unnecessary probes is to switch to passive mode in dense networks. However, such an approach would require adaptive beaconing from the APs to reduce the AP discovery latency which may end up increasing the energy consumption for the power saving devices.

Chapter 6

Managing PSM in Dense Networks

Energy management for wireless devices has been a core component of mobile computing research for years. However, current practices use solutions aimed at our expectations of sparse wireless networks from when such research started. When using current energy management solutions in the dense networks, the common result is that the devices are not able to send much data and, at the same time, their batteries drain at an increased rate that is not proportional to their successful data. The ultimate goal of our research is to rein in the energy consumption to match the device's data transmission. In doing so, we will show that we were able to reduce some of the contention in the network, ultimately increasing throughput, reducing delay and reducing energy consumption. Our research revealed that the existing power save mode (PSM) protocols are not necessarily broken. Rather, the biggest problem with the PSM implementations is that the protocols are agnostic of network conditions. In this paper, we propose `Skip-stop`, a PSM-based energy management approach that adapts to contention in dense networks by skipping AP polling during some beacon intervals and so limiting the number of devices active within each beacon interval. The ultimate goal of `Skip-stop` is to avoid introducing additional contention and energy consumption due to PSM, and to enable the devices to save energy when no useful communication can progress due to congestion.

6.1 Energy in Dense Networks

Rapid depletion of batteries in laptops and smartphones are very common phenomena when the devices are in dense networks. This happens even when the devices adopt energy-management protocols and are not actively transmitting data. The main problem for energy saving in dense networks is that, despite the research community's expectations [93], devices no longer spend most of their time idle listening. Instead, most energy overhead comes from retransmitting lost data and control packets due to the high contention and increased overhearing costs due to the high traffic load. While these effects may be intuitive, the impact on

existing duty-cycle-based power-saving protocols like IEEE 802.11 PSM [24] is even worse than expected. In this section, we describe and analyze how the existing IEEE standards and their proposed enhancements behave in dense networks.

6.1.1 The IEEE PSM Standards

The main goal of the IEEE 802.11 Power Save Mode (PSM) is to save energy wasted in idle listening by sleeping [24]. The packets sent to the sleeping devices are buffered at the AP and later retrieved from the AP when the devices wake up. By collecting and transmitting bursts of data, the devices minimize their idle awake times. The AP announces the presence of buffered packets at the beginning of each beacon interval for all of its associated PSM devices through the Traffic Indication Map (TIM) embedded in its beacons. The PSM devices adopting *Power Save Polling* (PSP) then retrieve their packets by polling the AP with one `PS-Poll` frame for each packet buffered at the AP. Devices can go back to sleep mode immediately after the AP indicates that there are no more buffered packets for that device. In sparse networks, where there is low contention, the devices go back to sleep after receiving their packets in bursts and hence minimize energy consumption. However, in dense networks, packets get arbitrarily delayed due to contending traffic that forces the devices to stay awake longer until they receive all packets from the AP.

Adaptive PSM avoids the overhead of `PS-Poll`s altogether by switching to constantly awake mode (CAM) once notified of pending packets by the AP and switching back to PSM after a *timeout*. The devices send `NULL-Data` frames to the AP when they switch to CAM mode and when they go back to sleep. While a short *timeout* aggressively saves energy by forcing the devices to sleep after a short idle time, a long *timeout* enables the devices to improve delay by receiving any near future transmissions within the same beacon interval. Devices can further improve energy-efficiency by increasing their *listen interval*, which determines how frequently they wake up. For example, with a *listen interval* of 1, the device wakes up every beacon interval, while with a *listen interval* of 3, the device wakes up every third beacon interval. However, a higher *listen interval* has the side effect of increasing delay. Hence, the choice of the parameter depends on the device's aggressiveness towards saving energy vs. reducing delay.

Since it is not easy to predict traffic patterns, all PSM implementations use fixed parameter values. Most devices wake up every beacon interval and use very short *timeout* values for their Adaptive PSM implementation. Naturally, the fixed values can not optimize the energy and delay when the traffic is dynamic

and unpredictable. When such devices are put in high density networks, congestion introduces additional complexities and makes these fixed values even more unsuitable. First, the short *timeout* may not be long enough to retrieve all buffered packets from the AP causing the AP buffer to grow and increasing congestion in the next beacon interval. Choosing a longer *timeout* does not resolve this problem since this may push the devices to postpone their time to go back to sleep, eventually resulting in increased energy consumption. A short *listen interval* value is also not appropriate for dense networks since it forces more devices to synchronize their channel accesses at the beginning of each beacon interval, resulting in contention spikes.

6.1.2 PSM Enhancements Targeting Delay

Several enhancements have been proposed over the years to further improve the delay performance of PSM devices. Although these approaches were not designed specifically for dense networks, we investigate whether any of these solutions is able to handle high density networks.

Two main factors are responsible for the increased delay experienced by PSM devices: inappropriate PSM parameter values and the presence of competing traffic in the network. Determining optimal values for PSM parameters requires foreknowledge about the traffic in the network. Since getting exact knowledge about traffic is not possible in dynamic networks, proposed solutions estimate future traffic patterns. Bounded Slow Down (BSD) [45] determines its *timeout* based on RTT estimation, and adapts the *listen interval* based on traffic history. Devices wake up more frequently when the traffic rate is high, while they slow down during low traffic. In dense networks, RTTs have large variance and thus determining the right *timeout* is difficult. This causes BSD to experience increased delay and energy consumption due to an inappropriate choice for *timeout*. Furthermore, the use of PSP causes a device to remain awake until it receives all buffered packets from the AP. In dense networks, this may take a long time, preventing the device from going back to sleep and making any use of its dynamic *listen interval*.

Dynamic *timeout* values are used by Smart-PSM [65] based on round trip time (RTT) estimation. Since, RTT may vary, further energy improvement is offered by Percy [28], which sets up an AP proxy that makes the RTTs uniform and enables the devices to choose a tight *timeout*. Such approaches do not work for dense networks since RTT is no longer the main factor determining delay.

The proposals [35, 68, 7, 36, 6] that optimize delay in the presence of background traffic are more likely to be suitable for managing congestion in dense networks. Traffic shaping [21, 76] is one approach of

handling congestion in the networks. By prioritizing the PSM traffic over CAM traffic, devices can separate the PSM and the CAM devices [86, 47, 72]. The AP transmits the PSM traffic first. The delay is reduced for the PSM devices since they are no longer affected by traffic from the CAM devices. However, there are two major problems with this approach. First, traffic from multiple PSM devices are still delayed by each other. Second, such scheduling is not fair to the CAM devices. Hence, such an approach is only appropriate when a single PSM device is contending with other CAM devices in each beacon interval. In dense networks where there will be many PSM devices along with many CAM devices, this approach is neither fair nor able to minimize delay.

Another approach to handle background traffic is to schedule the transmissions from the AP to the different devices such that each device gets its fair share [92]. Each device is allowed a separate time slot within the beacon interval to communicate with the AP. If the device knew exactly when the AP is going to send its packets, the device could wake up at that time and go back to sleep after receiving all of the packets. The challenge for these TDMA-based approaches is to let the devices know about their slots and to enable the devices to wakeup at that particular slot. Scheduled-PSM [37] passes the time slot information to the devices through special bits inside the beacons. Once devices get that information, they later wake up in their appropriate slot. However, passing information to the devices requires major changes to the IEEE standard and, hence, the solution is not readily deployable. A better solution is proposed by NAPman [68], which virtualizes the AP such that each device believes it has its own AP. These virtualized APs wake up at different times within the actual beacon interval allowing the devices to retrieve packets during their own time slot. Although this solution only requires modifying the AP without requiring modifications to the IEEE standard, such a solution is theoretical. In practice, NAPman can support only four PSM devices at best during a beacon interval, making the solution inappropriate for dense networks.

6.1.3 PSM Enhancements Targeting Energy

Overhearing is a major source of energy waste in dense networks due to the increased traffic. Solutions designed to reduce overhearing costs can certainly be used by PSM devices in dense networks to reduce their total energy consumption. During overhearing of a neighbor's packet, a device consumes almost the same energy as it would consume when receiving the packet itself. SNAF [18] reduces the cost of overhearing by allowing a device to sleep for the rest of the packet duration as soon as it identifies that the packet is

destined to some other node. Since, waking up the radio from sleep state is not free of cost in terms of both energy and time, instead of putting the device to sleep, Forced Idling [19] forces the radio to switch to the idle state when overhearing. Energy can be further improved by either switching the radio between on-off modes during duty cycling [8] instead of switching between on-sleep modes, or by running the devices at a lower clock frequency when traffic is not expected [93, 50]. The biggest limitation of these approaches is that all these approaches require modification of at least one of the followings: the packet structure, device drivers, and the IEEE standard. Hence, they are not readily deployable.

6.1.4 Road to Skip-stop

None of the PSM enhancements were designed to deal with high contention, hence they are not adequate to ensure survival in dense networks. However, the enhancements that target to improve energy-efficiency can be applied orthogonally to a dense network solution to achieve further energy-efficiency. To survive in dense networks, it is important to manage the PSM devices. If they are not properly controlled, the increased contention and packet drops from these devices may push the dense networks to an inoperable state. Additionally, their efforts to save energy may even result in increased energy consumption, when the network density and contention increase. Therefore, we next present our PSM-based solution to energy management in dense networks.

6.2 Skip-stop

`Skip-stop` is an energy management approach that adapts the use of PSM and its parameters based on local observations of network density and contention. Our approach (and the name of the protocol) is derived from solutions used in public transit systems to manage congestion during rush hours by having trains or buses skip chosen stops along their designated routes. The end result of `Skip-stop` is reduced contention while still allowing devices to save energy using PSM. Although a centralized approach might be able to select optimal parameters, it would require changes to AP software and perfect knowledge of traffic demands in the network. Therefore, `Skip-stop` is designed as a client-side local solution to manage PSM, which enables immediate and incremental deployment with no additional control overhead.

In the rest of this section, we describe the components of `Skip-stop` and its adaptive algorithm for setting PSM parameters based on network density.

6.2.1 Skip-stop Components

The goal of `Skip-stop` is to control the two main factors responsible for poor performance and increased energy consumption in dense networks: synchronized channel accesses by PSM devices that results in more contention and the combined effects of the increased traffic and delayed communication that causes the AP buffers for the PSM devices to grow beyond their limit and force more packet drops. To address these challenges, `Skip-stop` integrates two core components in each device: a *contention manager*, which makes sure that a PSM device does not introduce unnecessary contention, and a *buffer manager*, which pushes the device to retrieve its buffered packets from the AP before the buffers overflow.

The *contention manager* reduces the probability of synchronized channel accesses by adapting *listen interval* for the device based on observed contention in the network. If the contention is above a system defined threshold, *listen interval* is increased, which results in the device "skipping" some beacon intervals. When all devices employ `Skip-stop`, the number of awake devices is reduced for each beacon interval, allowing the network to reduce contention induced by synchronized channel accesses. The specific threshold can be defined by the system in terms of probability of collisions or packet drop ratio. We will be investigating the impact of different threshold values on `Skip-stop` as part of our future work.

The *buffer manager* aims to manage the problem of AP buffer overflow for the PSM device by trying to retrieve packets from the AP at a sufficient rate. In the presence of contention, the *contention manager* introduces delay by using a longer PSM *listen interval* and skipping some beacon intervals. Therefore, the buffers at the AP may fill up more than if the device retrieves packets every beacon interval. Therefore, the *buffer manager* sets the PSM *timeout* to help the device to drain the AP buffer each time it is awake.

The challenge for the *contention manager* and the *buffer manager* is to balance appropriate values for *timeout* and *listen interval*. Setting an arbitrarily long *listen interval* or *timeout* may make PSM worse. We next present `Skip-stop`'s adaptive algorithm to tune the PSM parameters according to the current network dynamics.

6.2.2 Dynamic Adaptation of PSM

`Skip-stop` must tune the PSM parameters according to the current network conditions since the network contention and density around a device change dynamically. While a long PSM *timeout* is always effective in reducing delay, such parameter setting can cause more energy waste due to idle listening in sparse networks.

Algorithm 2: Dynamic adaptation of PSM with Skip-stop

```
if slow_retrieval is SET then
    increment PSM timeout;
    if retx_ratio ≥ RETX_THRESHOLD then
        if PSM listen_interval < PSM_listen_interval_maximum then
            increment PSM listen_interval;
else
    if PSM timeout > PSM_timeout_minimum then
        decrement PSM timeout;
    if PSM listen_interval > 1 then
        decrement PSM listen_interval;
end
```

Hence, Skip-stop should use long *timeout*s in dense networks and shorter *timeout*s in sparse networks. The choice of an appropriate PSM *listen interval* is more critical. While skipping some beacon intervals may reduce contention due to synchronized channel accesses in dense networks, it has the side effect of delaying packet retrieval from the AP buffer, causing the buffer to grow and introducing more contention during the awake times. Hence, Skip-stop should increment *listen interval* only when desynchronizing the wakeups begets more benefit than delaying the communication.

Identifying the best PSM parameters requires knowledge of the actual network and traffic density. Since this information is not readily available to the devices, Skip-stop estimates network density in terms of two indicators of the current contention level: packets retrieved from the AP during the awake time and packet retransmissions. The AP buffer keeps on growing if the device is not able to retrieve all of its packets from the buffer during the awake interval. Although the device has no knowledge about the number of packets stored inside the AP buffer, it can still identify whether all packets have been retrieved or not by analyzing the more_data bit set inside each retrieved packet. Skip-stop marks this incident by turning on a flag called *slow_retrieval*. High packet retransmissions and *slow retrieval* indicate high contention, whereas the opposite indicate a low contention network.

The Skip-stop algorithm runs in rounds, each time acquiring a local estimation of the current contention level, and then deciding on how to adapt its PSM parameter values. The *buffer manager* increments the *timeout* value when *slow_retrieval* is set, and decrements the value when contention is low and the *slow_retrieval* flag is turned off (see Algorithm 2). Skip-stop allows the value to be decremented until it reaches a minimum allowable *timeout* value, PSM_timeout_minimum. Since the effect of a too

Algorithm 3: Desynchronizing PSM wakeups with Skip-stop

$current\_time$ = time( );
u = random(0, `PSM_listen_interval`);
$invoke\_time$ = $current\_time$ + u * `BEACON_INTERVAL`;
set new `PSM_listen_interval` and `PSM_timeout` at $invoke\_time$;

high *listen interval* can be detrimental, the *contention manager* increments the *listen interval* value only when the number of packet retransmissions is high and the percentage exceeds a system defined threshold, `RETX_THRESHOLD`. To avoid remaining at a high *listen interval* value unnecessarily, the *contention manager* decrements the value anytime *slow_retrieval* indicates a low contention network scenario.

6.2.3 Desynchronizing PSM Wakeups

The idea of `Skip-stop` is straight forward. If each device manages its PSM locally and limits the contention it introduces, then the additional network contention induced by all PSM devices will be under control. However, to balance out the level of contention across time, it is essential that different devices skip different stops. Otherwise, it will only aggravate contention, incurring additional delay and running the risk of AP buffer overrun. The *contention manager* in `Skip-stop` achieves this by waiting for a random time before setting its new *listen interval* value. Since, there are a total of *listen interval* stops within the device's wakeup period, the *contention manager* randomly chooses any of these stops as its first awake stop (see Algorithm 3). This desynchronizes the wakeup times of different devices with a very high probability.

By adapting the PSM devices, `Skip-stop` manages contention in dense networks and prevents the AP buffer from growing beyond limit, resulting in improved energy-efficiency as well as reduced delay. The effectiveness of `Skip-stop` along with its simplicity and inter-interoperability with existing PSM implementations make `Skip stop` an attractive solution which is readily deployable.

6.3 Evaluation

The goal of our evaluation is: (1) to demonstrate that the fixed PSM parameter value choices in the existing devices cannot handle varying network density, and (2) to evaluate `Skip-stop`'s ability to manage PSM by finding the best PSM setting.

Different smartphones implement PSM differently as decided by the vendor. Although the actual PSM

Devices	Timeout	Listen Interval
Nokia N900	200 msec	3
iPhone 4	95 msec	1
Android Nexus One	200 msec	1
HTC Hero	1500 msec	1

Table 6.1: PSM parameter values for smartphones

source code for any given device is not typically available, we reverse engineered the protocols by analyzing WireShark [26] traffic from different devices (i.e., iPhone, HTC Hero, Samsung Nexus S and Samsung Nexus Galaxy). The traces show that the iPhone uses Adaptive PSM and the rest of the devices use a combination of PSP and Adaptive PSM. These devices start in PSP and switch to Adaptive PSM only when they have data to transmit. When their *timeouts* expire, devices switch back to PSP. The parameter values used in these devices are also different. Table 6.3 lists these PSM parameter values. We implemented the associated PSM for each device and compared it to the use of Skip-stop using ns-2.35.

In our simulations, each Skip-stop round lasted for 20 sec. In a typical setting where a device wakes up every beacon interval, this would allow the device to base its decision about the current PSM parameter values on information collected over 200 awake intervals. Skip-stop sets the PSM_timeout_minimum at 20 msec, whereas allows the *listen interval* to rise until 10. Each time Skip-stop increments the value of of *timeout*, it increments the value by 20 msec, whereas the granularity for changing *listen interval* is set to 1. Our simulations use a RETX_THRESHOLD value of 50%. Our exhaustive simulations show that using a lower RETX_THRESHOLD value results in unnecessary increment of *listen interval*.

We present our simulation results in two steps. First, we show that the fixed PSM parameter settings are not appropriate to handle the various network and traffic densities expected by the devices. Results include PSM behavior in the following smartphones: Android Nexus One, iPhone 4, iPhone 3G, and HTC Hero. In the second stage, we evaluate Skip-stop's ability to manage contention by dynamically tuning the PSM parameters.

6.3.1 Performance Metrics

For the exhaustive analyses with fixed PSM settings, we evaluated performance using *throughput*, *delay* and *packet drop ratio*. The *throughput* is an aggregate over all devices and hence captures network's ability to handle more PSM devices. A lower packet drop ratio is also an indication of better management of

contention in dense networks, whereas increased packet drops indicate the opposite. Delay is particularly important in sparse networks, since a device should not have to wait long to send its data. Finally, to measure energy-efficiency, we evaluate *energy per bit*, which reflects the actual cost to the device for the data it is sending. Higher energy per bit indicates that the devices is spending more of its energy dealing with contention and poor parameter choices instead of on data. Since `Skip-stop` prevents network breakdown by limiting the contention induced by PSM, we measure `Skip-stop`'s effectiveness by evaluating the same metrics as PSM evaluation.

6.3.2 Network and Traffic

To evaluate the impact of the different approaches on a single AP, the simulated network has randomly placed devices in a 50 m$\times$ 50 m space. The number of devices in the network is varied from 10 to 150, all connecting to one AP. Although the high end is beyond the recommended number of devices per AP, we have observed even higher number of devices in traces of dense networks.

Traffic from each device was set to closely approximate real Internet traffic patterns for smartphone users. According Internet measurement studies [32, 34, 17], both flow sizes and inter-arrival times vary depending on the applications running on the devices. In our simulations, 80% of all flows had flows less than 1 KB. The inter-arrival was also chosen according to the same measurements which suggest that 80% of the flows arrive within 1 sec and the rest have a long tail at 18 sec.

For all simulations, devices join the network at random times within the first 60 seconds of the simulated time. Data was collected for a 30 minute long simulation. Results presented in this section are obtained from an average of multiple simulation runs.

6.3.3 PSM Behavior with Fixed Parameters

Two main factors determine the behavior of PSM in dense networks—growing AP buffers and synchronized channel accesses. To demonstrate the impact of the growing AP buffer, we evaluated the performance of PSM with a fixed *listen interval* value of 1 and with various *timeout* settings—25, 95, 200 and 1500 msec as used by the different smartphone vendors (see Table 6.3). We also investigate the impact of synchronized channel accesses by evaluating their achievable performance if these devices were allowed to use longer *listen interval*s.

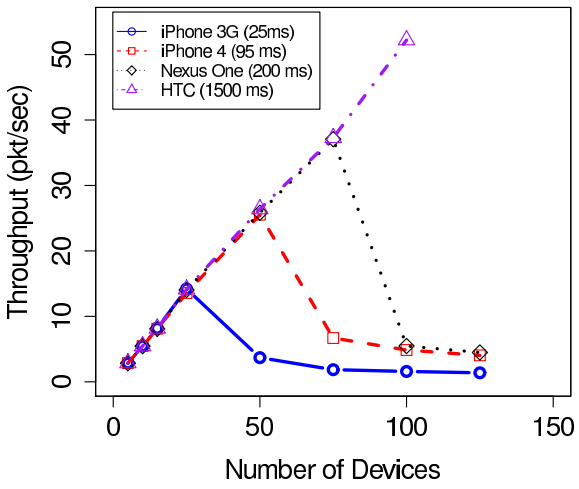

Figure 6.1: Throughput with fixed *listen interval*

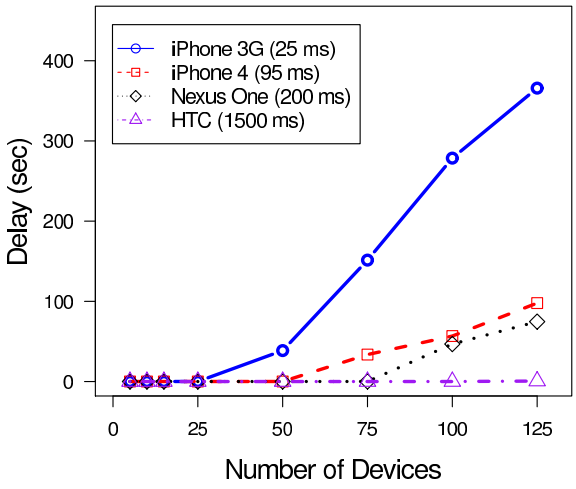

Figure 6.2: Delay with fixed *listen interval*

Impact of PSM timeout

The *throughput* trend of PSM devices in dense networks is completely different than what we typically expect in sparse networks where contention is low. In a "well managed" network, the *throughput* is expected to rise until a certain point and remain at that value since the network saturates as more devices join the network. However, the PSM devices in dense networks experience exponential decrease in *throughput* as the network starts thrashing due to increased contention. It is clear from Figure 6.1 that the PSM device's ability to handle contention is directly dependent on its *timeout* value. PSM devices already induce contention due to synchronized channel accesses. A shorter PSM *timeout* value increases contention even more by not allowing the device to retrieve all of its packets from the AP buffer and forcing them to contend for channel access again in the next beacon interval. For this reason, `iPhone 3G`, having a *timeout* of 25 msec, starts its steep decrease at a very low density of 50 devices. In contrast, a longer *timeout* value allows the device to retrieve more packets from the AP buffer with a higher probability, reducing contention for the next beacon interval. As longer *timeout*s are used, the network breakdown occurs at higher density. We observed the best *throughput* performance in dense networks with the `HTC` phones among all the other PSM settings that we evaluated. Having a *timeout* of 1500 msec, the `HTC` devices can handle network density upto 150 devices, after which they start thrashing the network.

The longer *timeout* also results in immense improvements in *delay* and *packet drop ratio* since the AP buffer is now well managed and the contention is under control. The dangers of using short *timeout* values

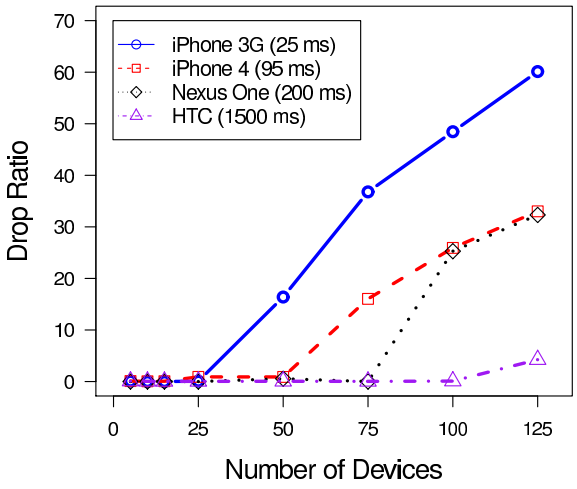

Figure 6.3: Drop ratio with fixed *listen interval*

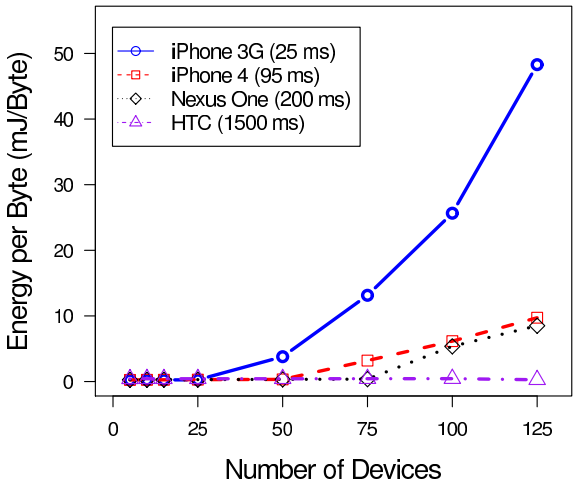

Figure 6.4: Energy per byte with fixed *listen interval*

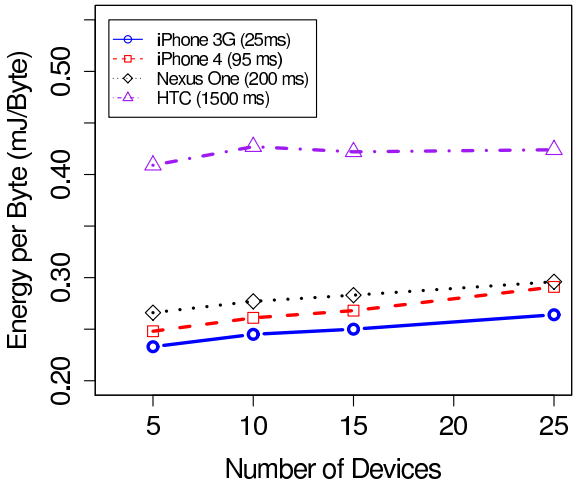

Figure 6.5: Energy per byte with fixed *listen interval*: sparse networks

in dense networks are evident from the results (see Figure 6.2) as `iPhones` experience an *average delay* of 366 sec compared to the 31 msec delay of `HTC` phones in a network with 125 devices. PSM's ability to manage contention is better demonstrated by observing the *packet drop ratio* for the different phones in varying network densities. The devices start dropping packets at higher network density when they use longer *timeout*s. While `iPhone` starts dropping 15% of its packets at a density of 50 devices, the `Nexus One` drops that many packets beyond 75 device density (see Figure 6.3). The best performance is achieved by the `HTC` phones which can maintain their drop rate to below 1% even in networks with 125 devices.

The biggest problem with PSM devices in dense networks is that most devices have very poor commu-

nication quality, but they exhaust their energy at a very fast rate. Typically, energy consumption is expected to reduce in sparse networks with shorter *timeout*s since devices can go back to sleep early. To save energy, most smartphone vendors choose short *timeout* values. However, in dense networks, the trends are completely opposite. The energy performance in these networks are dominated by the number of successful packet transmissions. While the HTC phones consume a constant 0.4 mJ per byte for data transmissions, iPhones experience an exponential increase in energy consumption when the density increases (see Figure 6.4). At the density of 125 devices, iPhone consumes 100 times higher *energy per byte* due to delay and packet drops caused by increased contention.

All results indicate that longer *timeout*s result in better communication quality—increased throughput, reduced delay and packet drops, and improved energy-efficiency in dense networks and suggest using HTC phones in dense networks. But one must make sure that the longer *timeout* does not result in poor performance or increased energy consumption in sparse networks which the devices typically expect. In sparse networks, devices experience similar *throughput* and very low *packet drop rate*s for all *timeout*s. The delay trends are similar to those of the dense networks providing improved *delay* for longer *timeouts*. However, all these improvements by setting long *timeout*s are achieved at the cost of very high increase in energy consumptions for sparse networks. When the contention is low and the devices are idle most of the time, longer *timeout*s results in energy waste due to idle listening. The HTC phones consume at least 4 times more *energy per byte* than the iPhones (see Figure 6.5). Since a device will be in sparse networks most of the time, such high energy consumption indicates that an HTC phone will have a quarter of the battery lifetime compared to an iPhone. Hence, devices must set a short *timeout* in sparse networks, whereas longer *timeout* in dense networks.

Impact of PSM listen interval

One of the main reasons for "PSM breakdown" in dense networks is the high number of synchronized channel accesses. By using a higher *listen interval*, PSM can reduce the number of devices active within each beacon interval. However, this can effectively improve the quality of communication only when the performance is dominated by the synchronized channel accesses. iPhone 3G will achieve the maximum benefit from an increased *listen interval* since the short *timeout* results in a growing AP buffer that increases the number of contending devices in subsequent beacon intervals. iPhones can reduce their drop rates

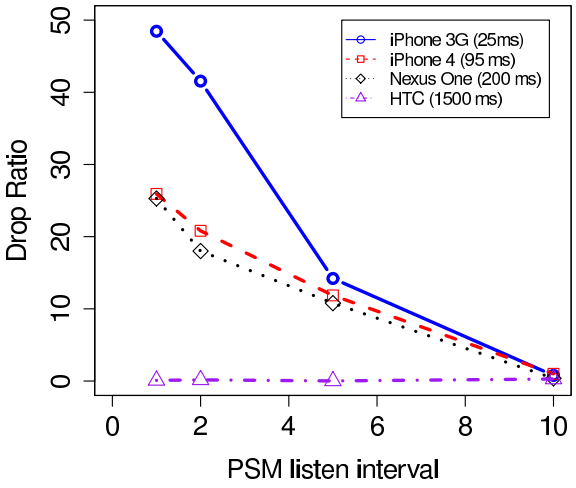

Figure 6.6: Drop ratio with fixed *timeout* (network density=100 devices)

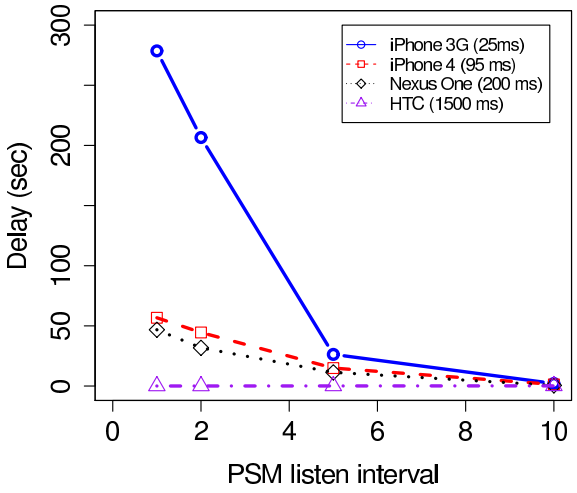

Figure 6.7: Delay with fixed *timeout* (network density=100 devices)

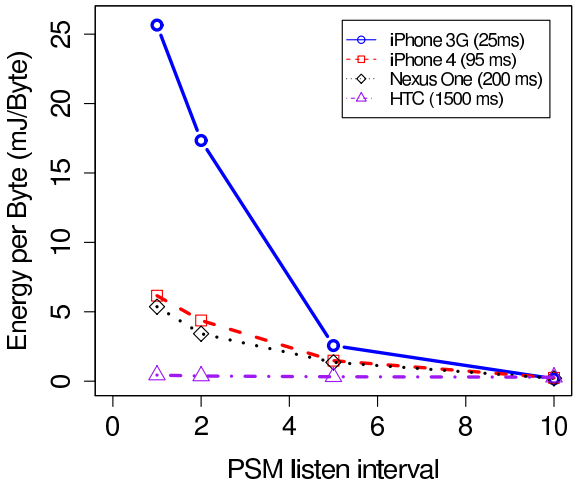

Figure 6.8: Energy per byte with fixed *timeout* (network density=100 devices)

by 14% if they wake up every other beacon interval instead of waking up for all beacons (see Figure 6.6). The drop rate further reduces by 70% if the `iPhones` would have set their *listen interval* to 5. Limiting the synchronized accesses also results in immense improvements in delay. With the *listen interval* set to 5, `iPhones` can reduce the average delay by 99% compared to the setting when they wake up every beacon interval (see Figure 6.7). Finally, the increased probability of successful packet transmissions and the reduced delay combined with the increased sleep time for each device due to delayed wakeup improve the energy-efficiency. Devices consume 10 times lower *energy per byte* than the existing setup in which the *listen interval* is set to 1 (see Figure 6.8). Hence, PSM devices with short *timeouts*, e.g., `iPhones`, will

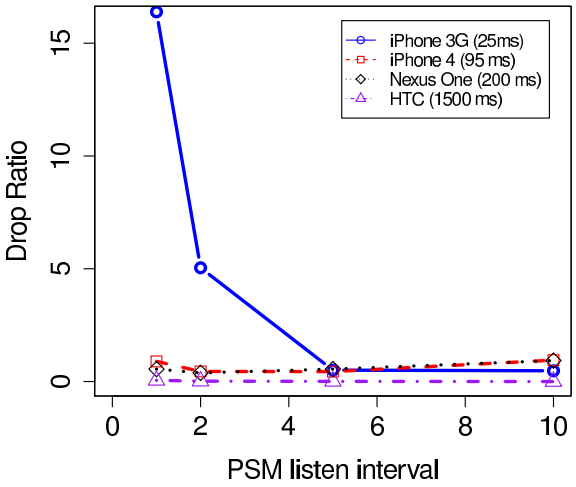

Figure 6.9: Drop ratio with fixed *timeout* (network density=50 devices)

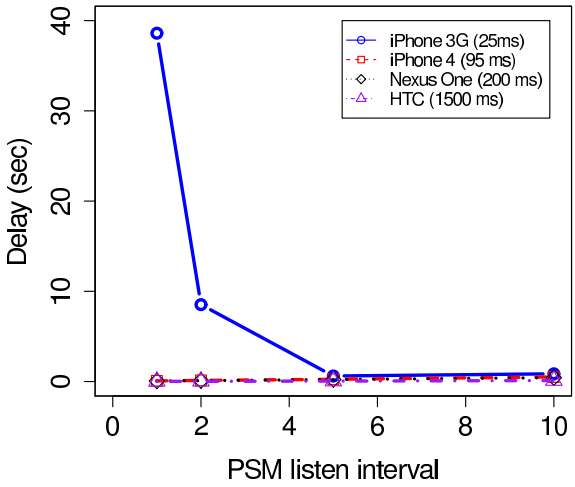

Figure 6.10: Delay with fixed *timeout* (network density=50 devices)

definitely benefit from using longer *listen interval*s in dense networks.

For the devices with longer *timeouts*, e.g., `iPhone 4`, `Nexus One`, `HTC`, the percentage improvements are lower compared to `iPhone 3G` when the *listen interval*s are increased. As discussed earlier in Section 6.3.3, longer *timeout* devices are better adept in limiting the synchronized channel accesses, hence they provide fewer opportunities to improve communication by increasing *listen interval*. With the *listen interval* set to 5, both `iPhone 4` and `Nexus One` reach almost similar low values for their drop rates, delay and energy consumption (see Figure 6.6, 6.7 and 6.8). Although both of these phones show even better performance when the *listen interval* is increased to 10, it is evident from the performance of `HTC` that always increasing the *listen interval* does not necessarily provide better quality of communication. The `HTC` phones already maintain a very low drop rate of below 1% when they wakeup for every beacon in dense networks (see Figure 6.6). The 1500 msec *timeout* ensures that devices clear out the AP buffer when they wake up, hence they do not increase the number of synchronized accesses in the subsequent beacon intervals. In this situation, delaying the wakeups can only result in exponential increase in delay. The average *delay* doubles when the `HTC` phones run with a *listen interval* of 5, and increases by 350 times when the *listen interval* is increased to 10 (see Figure 6.7). Hence, the best *listen interval* to use in dense networks depends on the value of PSM *timeout*.

If a fixed *listen interval* were to use by each phone type, one must make sure that the setting can improve the quality of communication by limiting the synchronized channel accesses in dense networks and it must

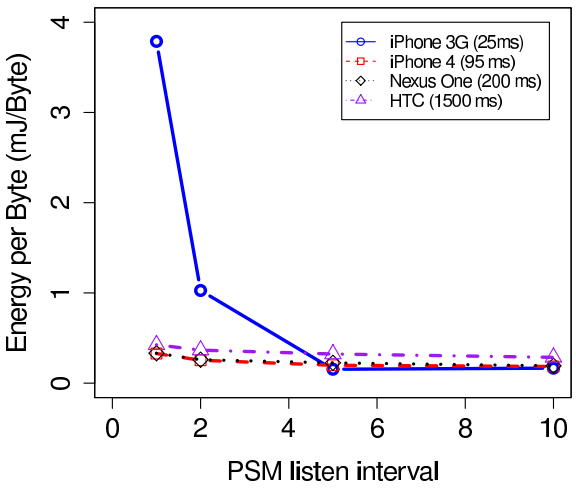

Figure 6.11: Energy per byte with fixed *timeout* (network density=50 devices)

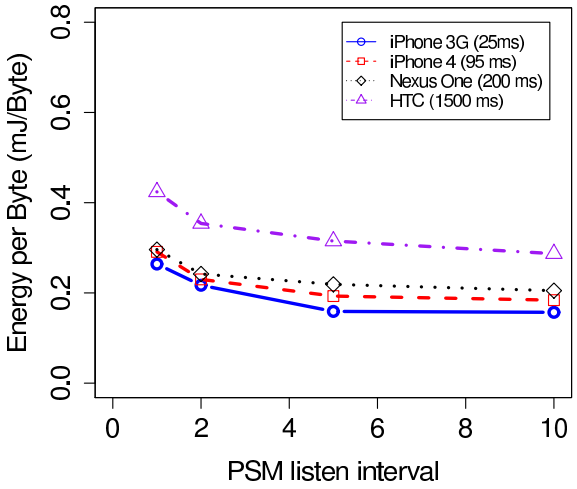

Figure 6.12: Energy per byte with fixed *timeout* (network density=25 devices)

not deteriorate the performance in sparse networks. Only `iPhone 3G` benefits from using higher *listen interval* when the network consists of 50 devices. The drop rate reduces by 69% when the devices wakeup every other interval and drops below 1% when the *listen interval* is set to 5 (see Figure 6.9). Similar improvements are achieved in terms of delay (see Figure 6.10) and energy-efficiency (see Figure 6.11). However, for the rest of the phones, increasing the *listen interval* at this network density resulted in huge increase in delay compared to the small improvements in energy-efficiency. For example, when `Nexus One` sets its *listen interval* to 5, the energy-efficiency is improved by 32% at the cost of increasing the delay by 274 times (see Figure 6.10 and 6.11). The same trend is observed for all devices including `iPhone 3G` when the network has fewer than 50 devices. Even the improvements in energy-efficiency for all devices become insignificant when the *listen interval* is increased beyond 5 (see Figure 6.12). Hence, the huge increase in delay dominates over the small improvements in energy-efficiency and increasing the *listen interval* in sparse networks is no longer beneficial.

To summarize, although higher *listen interval*s provide better quality of communication for most devices in dense networks, the delayed wakeups severely affect the delay in sparse networks. An increased *listen interval* is only desirable when the performance is affected by high number of synchronized channel accesses. Since synchronized access is determined by the number of devices in the network and their PSM *timeout*s, the best *listen interval* must vary according to the *timeout* and the network density.

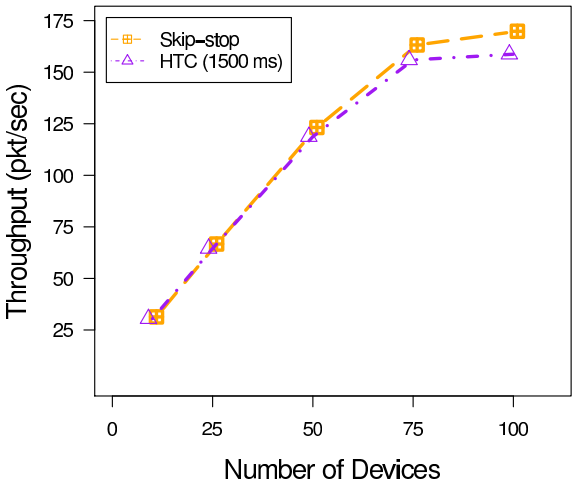

Figure 6.13: Throughput with Skip-stop

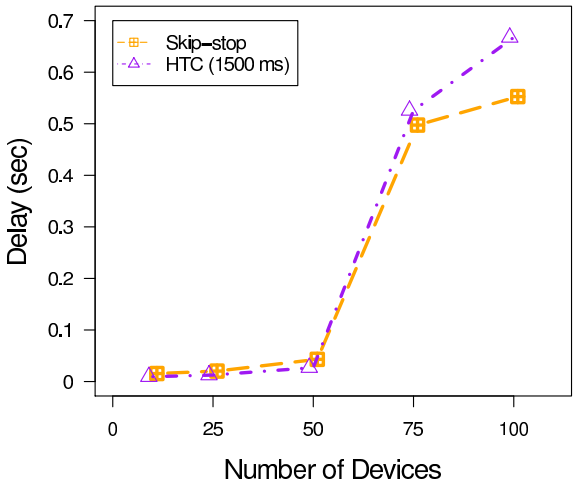

Figure 6.14: Delay with Skip-stop

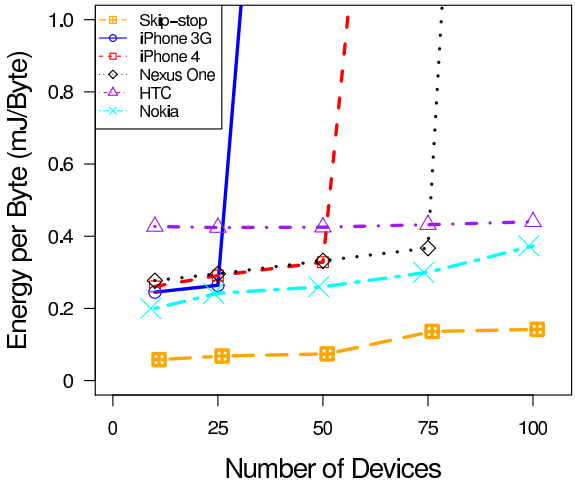

Figure 6.15: Energy per byte with Skip-stop

6.3.4 Effectiveness of Skip-stop

By dynamically adapting the PSM parameters according to the contention in the networks, `Skip-stop` successfully copes with high contention and achieves immense improvements in the quality of communication. `Skip-stop` starts with the `HTC` setting and later modifies the parameters according to the dynamic changes in the network density and traffic. The success of `Skip-stop` is evident as the devices reduce delay by 17% (see Figure 6.14) and improve throughput by 7% (see Figure 6.13) when compared to `HTC`, the device with the best performance at a density of 100 devices. The improvements when using `Skip-stop` was possible because of the flexible PSM parameters. Our simulations showed that `Skip-stop` raised the PSM *listen interval* upto a maximum value of 3 and switched to waking up every third beacon interval

during a short period of high contention. To facilitate faster retrieval of data from the AP buffer, the *timeout* was also increased upto a maximum value of 1600 msec.

While the increased parameter values enable the devices to better cope with high contention, the flexibility of switching back to using shorter *timeout* and *listen interval* allows the devices to reduce their energy consumption during low contention. Energy-efficiency in sparse networks is crucial and it was the biggest barrier against choosing `HTC` as an ideal solution. Among the existing devices, `Nokia` and `iPhone 3G` are the best in terms of energy-efficiency during low contention (see Figure 6.15). In a sparse network with only 10 devices, `Skip-stop` improves the energy-efficiency by 71% compared to `Nokia` and by 76% when compared to `iPhone 3G`.

To summarize, `Skip-stop` improves both the quality of communication and the energy-efficiency in dense networks by managing contention. By being adaptive to network density, it also achieves significant improvements in energy-efficiency in sparse networks. Hence, `Skip-stop` is an ideal energy-management approach that enables the PSM devices to survive in dense networks.

6.4 Conclusions and Future Directions

The contribution of this work is to identify the devastating affect of PSM in dense networks and to devise a simple solution to prevent the network break down. We designed `Skip-stop`, an energy management approach to control PSM in dense networks to prevent it from introducing additional spikes of contention. It is important to note that `Skip-stop` does not solve the problem of overloading in dense networks. Instead, it mitigates the problem of contention in an overloaded network allowing devices to operate in such networks without overly taxing their energy resources. To do so, `Skip-stop` dynamically adapts to the density of traffic and devices in the network by tuning two PSM parameters: *listen interval* and *timeout* appropriately. Our results show that setting appropriate parameter values for the current network conditions not only saves energy in both sparse and dense networks, but also improves delay and throughput in dense networks.

Chapter 7

Conclusions and Future Directions

The future wireless networks will be very dense. While many wireless networks are still sparse and lightly loaded, we are seeing more and more hot-spots where the wireless network is simply overwhelmed by devices and traffic. Although there are many research efforts to solve the load-balancing problem of which devices should connect to which access points, in busy environments where many people converge, most access points will still be overloaded. Hence, these energy-constrained high density networks have very special challenges. The challenges start at an early stage of communication when the devices decide to join a network and so actively search each channel with probe packets to obtain nearby AP information. In dense networks, this initiates cycles of probes from all devices and brings down the entire network. The challenges continue even after the devices associate with an AP. They struggle to maintain a good quality of communication without spending too much energy in the presence of high contention from background traffic. The limited energy resources of the devices complicate the entire process. If the energy-saving protocols are not designed properly, instead of improving communication and achieving better energy-efficiency, they raise contention even more deteriorating the communication quality by causing packet drops, increasing delay and reducing throughput, and finally increasing their energy consumption.

In our thesis, we have focused on solving some of these problems. Each solution leveraged the inherent characteristics of a dense networks such as increased connectivity, frequent traffic and redundancy to improve communication and achieve energy-efficiency. For example, NPM, Any-MAC and Jam-Buster improved energy, delay and security of WSN communication by enabling collaboration between neighbors. Dynamo-Probing and Skip-stop focused on managing contention in an effort to ensure survival in dense WLAN. While each component is essential to improve energy-efficiency and communication in dense networks, a comprehensive solution would require us to combine all of them together. The first step towards achieving this goal would be to identify the core problems by study and rigorous analysis of the empirical data collected from dense networks. From our research on dense networks, we have learned that the key

to survive in high contention is to embrace the density. Hence, by exploiting every opportunities these networks provide, we can design a comprehensive and practical solution which would enable survival in dense networks.

References

[1] Boeing's 'virtual fence' on mexico border is halted. http://seattletimes.nwsource.com/html/nationworld/2011362012_border17.html.

[2] Building a border: Part 2. http://www.denverpost.com/fortressamerica/ci_5356695.

[3] Students develop sensor network to monitor forest. http://www.informationweek.com/news/190301069.

[4] Wildlife net-gamekeepers using sensor network. http://dl.acm.org/citation.cfm?id=1326269.

[5] Cisco visual networking index: Global mobile data traffic forecast update, 2011-2016. *Cisco white paper*, 2012.

[6] P. Agrawal, A. Kumar, J. Kuri, M. K. Panda, V. Navda, and R. Ramjee. Opsm-opportunistic power save mode for infrastructure ieee 802.11 wlan. In *Communications Workshops (ICC), 2010 IEEE International Conference on*. IEEE.

[7] P. Agrawal, A. Kumar, J. Kuri, M. K. Panda, V. Navda, R. Ramjee, and V. Padmanabhani. Analytical models for energy consumption in infrastructure wlan stas carrying tcp traffic. In *Communication Systems and Networks (COMSNETS), 2010 Second IEEE International Conference on*, 2010.

[8] G. Anastasi, M. Conti, E. Gregori, and A. Passarella. 802.11 power-saving mode for mobile computing in wi-fi hotspots: limitations, enhancements and open issues. *Wireless Networks*, 2008.

[9] J. Andrews, H. Claussen, M. Dohler, S. Rangan, and M. Reed. Femtocells: Past, present, and future. *In Proceedings of IEEE Journal on Sel. Areas in Communication*, 2012.

[10] F. Ashraf, R. Crepaldi, and R. Kravets. Know Your Neighborhood: A Strategy for Energy-efficient Communication. In *MASS*, 2010.

[11] F. Ashraf, R. Crepaldi, and R. Kravets. Synchronization vs. Signaling: Energy-Efficient Coordination in WSN. In *Wireless Mesh Networks (WIMESH 2010), 2010 Fifth IEEE Workshop on*, 2010.

[12] F. Ashraf, Y.-C. Hu, and R. Kravets. Demo: bankrupting the jammer. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, 2011.

[13] F. Ashraf, Y.-C. Hu, and R. H. Kravets. Bankrupting the jammer. In *Sensor, Mesh and Ad Hoc Communications and Networks (SECON), 2011 8th Annual IEEE Communications Society Conference on*, 2011.

[14] F. Ashraf and R. Kravets. Poster abstract: neighborhood-based power management. *ACM SIGMOBILE Mobile Computing and Communications Review*, 2010.

[15] F. Ashraf, R. H. Kravets, and N. H. Vaidya. Exploiting routing redundancy using mac layer anycast to improve delay in wsn. *ACM SIGMOBILE Mobile Computing and Communications Review*, 2010.

[16] F. Ashraf, N. H. Vaidya, and R. H. Kravets. Any-mac: Extending any asynchronous mac with anycast to improve delay in wsn. In *Sensor, Mesh and Ad Hoc Communications and Networks (SECON), 2011 8th Annual IEEE Communications Society Conference on*, 2011.

[17] B. Augustin and A. Mellouk. On traffic patterns of http applications. In *GLOBECOM*, 2011.

[18] B. Balaji, B. R. Tamma, and B. Manoj. A novel power saving strategy for greening ieee 802.11 based wireless networks. In *Proceedings of IEEE Conference on Global Telecommunications*, 2010.

[19] S. Biswas and S. Datta. Reducing overhearing energy in 802.11 networks by low-power interface idling. In *Proceedings of IEEE International Conference on Performance, Computing, and Communications*, 2004.

[20] M. Buettner, G. Yee, E. Anderson, and R. Han. X-MAC: a short preamble MAC protocol for duty-cycled wireless sensor networks. In *SenSys*, 2006.

[21] S. Chandra and A. Vahdat. Application-specific network management for energy-aware streaming of popular multimedia formats. In *USENIX Annual technical conference*, 2002.

[22] X. Chen and D. Qiao. Hand: fast handoff with null dwell time for ieee 802.11 networks. In *INFOCOM*, 2010.

[23] V. Chintala and Q. Zeng. Novel mac layer handoff schemes for ieee 802.11 wireless lans. In *Proceedings of IEEE Wireless Communications and Networking Conference*, 2007.

[24] J. Choi, Y. Ko, and J. Kim. Enhanced power saving scheme for IEEE 802.11 DCF based wireless networks. *Lecture notes in computer science*, 2003.

[25] R. Choudhury and N. Vaidya. Mac-layer anycasting in ad hoc networks. *ACM SIGCOMM Computer Communication Review*, 2004.

[26] G. Combs et al. Wireshark-network protocol analyzer. *Version 3.5.1*, 2012.

[27] J. Deng, Y. Han, W. Heinzelman, and P. Varshney. Scheduling sleeping nodes in high density cluster-based sensor networks. *Mobile Networks and Applications*, 10(6):825–835, 2005.

[28] N. Ding, A. Pathak, D. Koutsonikolas, C. Shepard, Y. Hu, and L. Zhong. Realizing the full potential of psm using proxying. In *Proceedings of IEEE INFOCOM*, 2012.

[29] S. Du, A. Saha, and D. Johnson. RMAC: A routing-enhanced duty-cycle mac protocol for wireless sensor networks. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, 2007.

[30] A. El-Hoiydi and J. Decotignie. WiseMAC: an ultra low power MAC protocol for the downlink of infrastructure wireless sensor networks. In *iscc*, 2004.

[31] J. Eriksson, H. Balakrishnan, and S. Madden. Cabernet: vehicular content delivery using wifi. 2008.

[32] H. Falaki, D. Lymberopoulos, R. Mahajan, S. Kandula, and D. Estrin. A first look at traffic on smartphones. In *IMC*, 2010.

[33] H. Fusler, J. Widmer, M. Kasemann, M. Mauve, and H. Hartenstein. Contention-based forwarding for mobile ad hoc networks* 1. *Ad Hoc Networks*, 2003.

[34] S. Gebert, R. Pries, D. Schlosser, and K. Heck. Internet access traffic measurement and analysis. *Traffic Monitoring and Analysis*, 2012.

[35] Y. He and R. Yuan. A novel scheduled power saving mechanism for 802.11 wireless lans. *Mobile Computing, IEEE Transactions on*, 2009.

[36] Y. He, R. Yuan, X. Ma, and J. Li. Analysis of the impact of background traffic on the performance of 802.11 power saving mechanism. *IEEE Communications Letters*, 2009.

[37] Y. He, R. Yuan, X. Ma, J. Li, and C. Wang. Scheduled psm for minimizing energy in wireless lans. In *Proceedings of IEEE International Conference on Network Protocols*, 2007.

[38] C. Hu, R. Zheng, J. Hou, and L. Sha. A microscopic study of power management in IEEE 802.11 wireless networks. *International Journal of Wireless and Mobile Computing*, 2006.

[39] H. Huang, T. Huang, N. Chilamkurti, R. Cheng, and C. Shieh. Adaptive forward error correction with cognitive technology mechanism for video streaming over wireless networks. In *3CA*, 2010.

[40] E. Hyytiä and J. Virtamo. On traffic load distribution and load balancing in dense wireless multihop networks. *EURASIP Journal on Wireless Communications and Networking*, 2007.

[41] T. Instruments. CC2420 Datasheet. *Reference SWRS041B, March*, 2008.

[42] F. J., J. Whiteaker, A. Cuellar Amrod, and J. Woodhams. Wireless lan design guide for high density client environments in higher education. *Cisco Design Guide*, 2011.

[43] S. Jain and S. Das. Exploiting path diversity in the link layer in wireless ad hoc networks. *Ad Hoc Networks*, 2008.

[44] B. Jang, J. Lim, and M. Sichitiu. AS-MAC: An asynchronous scheduled MAC protocol for wireless sensor networks. In *5th IEEE International Conference on Mobile Ad Hoc and Sensor Systems, 2008. MASS 2008*, 2008.

[45] R. Krashinsky and H. Balakrishnan. Minimizing energy for wireless web access with bounded slowdown. In *Proceedings of the ACM 8th annual international conference on Mobile computing and networking*, 2002.

[46] Y. Law, L. van Hoesel, J. Doumen, P. Hartel, and P. Havinga. Energy-efficient link-layer jamming attacks against wireless sensor network MAC protocols. In *Proceedings of the 3rd ACM workshop on Security of ad hoc and sensor networks*, pages 76–88. ACM, 2005.

[47] H.-P. Lin, S.-C. Huang, and R.-H. Jan. A power-saving scheduling for infrastructure-mode 802.11 wireless lans. *Computer communications*, 2006.

[48] S. Lin and D. Costello. *Error control coding*. Prentice-Hall Englewood Cliffs, NJ, 1983.

[49] S. Liu, K. Fan, and P. Sinha. CMAC: An energy-efficient MAC layer protocol using convergent packet forwarding for wireless sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, 2009.

[50] F. Lu, G. M. Voelker, and A. C. Snoeren. Slomo: Downclocking wifi communication. In *Proceedings of NSDI*, 2013.

[51] G. Lu, B. Krishnamachari, and C. Raghavendra. An adaptive energy-efficient and low-latency MAC for data gathering in wireless sensor networks. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, 2004.

[52] D. MacKay. Fountain codes. In *IEEE Communications*, 2005.

[53] J. Manweiler and R. Roy Choudhury. Avoiding the rush hours: Wifi energy management via traffic isolation. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, 2011.

[54] S. Matta. Broadcast probe responses. http://tinyurl.com/a8rp7qx.

[55] A. Mishra, M. Shin, and W. Arbaugh. An empirical analysis of the ieee 802.11 mac layer handoff process. *In Proceedings of SIGCOMM*, 2003.

[56] A. Mishra, M. Shin, and W. Arbaush. Context caching using neighbor graphs for fast handoffs in a wireless network. In *Proceedings of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies INFOCOM 2004.*, 2004.

[57] A. Mishra, M. Shin, N. Petroni Jr, T. Clancy, and W. Arbaugh. Proactive key distribution using neighbor graphs. *In Proceedings of IEEE Wireless Communications*, 2004.

[58] S. Pack, J. Choi, T. Kwon, and Y. Choi. Fast-handoff support in ieee 802.11 wireless networks. *In Proceedings of IEEE Communications Surveys and Tutorials*, 2007.

[59] S. Pack, H. Jung, T. Kwon, and Y. Choi. Snc: a selective neighbor caching scheme for fast handoff in ieee 802.11 wireless networks. *In Proceedings of ACM SIGMOBILE Mobile Computing and Communications Review*, 2005.

[60] G. Park, K. Ryu, and J. Kwak. Triggering the broadcast probe response. http://tinyurl.com/amzkxlc.

[61] V. Paruchuri, S. Basavaraju, A. Durresi, R. Kannan, and S. Iyengar. Random asynchronous wakeup protocol for sensor networks. 2004.

[62] J. Pita, M. Jain, J. Marecki, F. Ordóñez, C. Portway, M. Tambe, C. Western, P. Paruchuri, and S. Kraus. Deployed armor protection: the application of a game theoretic model for security at the los angeles international airport. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: industrial track*, 2008.

[63] J. Pita, M. Jain, F. Ordónez, C. Portway, M. Tambe, C. Western, P. Paruchuri, and S. Kraus. Using game theory for los angeles airport security. *AI Magazine*, 2009.

[64] J. Polastre, J. L. Hill, and D. E. Culler. Versatile low power media access for wireless sensor networks. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems, SenSys 2004, Baltimore, MD, USA, November 3-5, 2004*. ACM, 2004.

[65] D. Qiao and K. Shin. Smart power-saving mode for ieee 802.11 wireless lans. In *Proceedings of IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, 2005.

[66] I. Ramani and S. Savage. Syncscan: practical fast handoff for 802.11 infrastructure networks. In *Proceedings of 24th Annual Joint Conference of the IEEE Computer and Communications Societies INFOCOM*, 2005.

[67] T. Rappaport. *Wireless communications*. Prentice Hall PTR, 2002.

[68] E. Rozner, V. Navda, R. Ramjee, and S. Rayanchu. Napman: network-assisted power management for wifi devices. In *Proceedings of the ACM 8th international conference on Mobile systems, applications, and services*, 2010.

[69] M. Shin, A. Mishra, and W. Arbaugh. Improving the latency of 802.11 hand-offs using neighbor graphs. In *MASS*, 2004.

[70] M. Shin, A. Mishra, and W. Arbaugh. Improving the latency of 802.11 hand-offs using neighbor graphs. In *Proceedings of the 2nd International Conference on Mobile systems, Applications, and Services*, 2004.

[71] D. Soldani and S. Dixit. Wireless relays for broadband access [radio communications series]. *In Proceedings of IEEE Communications Magazine*, 2008.

[72] J. A. Stine and G. De Veciana. A comprehensive energy conservation solution for mobile ad hoc networks. In *Communications, 2002. ICC 2002. IEEE International Conference on*. IEEE, 2002.

[73] A. Subramanian, H. Gupta, S. Das, and J. Cao. Minimum interference channel assignment in multiradio wireless mesh networks. *In Proceedings of IEEE Transactions on Mobile Computing*, 2008.

[74] Y. Sun, S. Du, O. Gurewitz, and D. Johnson. DW-MAC: a low latency, energy efficient demand-wakeup MAC protocol for wireless sensor networks. In *MobiHoc*, pages 53–62. ACM, 2008.

[75] Y. Sun, O. Gurewitz, and D. Johnson. RI-MAC: a receiver-initiated asynchronous duty cycle MAC protocol for dynamic traffic loads in wireless sensor networks. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*. ACM, 2008.

[76] E. Tan, L. Guo, S. Chen, and X. Zhang. Psm-throttling: Minimizing energy consumption for bulk data communications in wlans. In *Proceedings of IEEE International Conference on Network Protocols*, 2007.

[77] H. Tan and M. Chan. A^2-MAC: An Adaptive, Anycast MAC Protocol for Wireless Sensor Networks. In *Wireless Communications and Networking Conference (WCNC), 2010 IEEE*, 2010.

[78] L. Tang, Y. Sun, O. Gurewitz, and D. Johnson. PW-MAC: An Energy-Efficient Predictive-Wakeup MAC Protocol for Wireless Sensor Networks. 2011.

[79] V. Tarokh, N. Seshadri, and A. Calderbank. Space-time codes for high data rate wireless communication: Performance criterion and code construction. *IEEE Transactions on Information Theory*, 1998.

[80] J. Teng, C. Xu, W. Jia, and D. Xuan. D-scan: Enabling fast and smooth handoffs in ap-dense 802.11 wireless networks. In *Proceedings of IEEE INFOCOM*, 2009.

[81] D. Thuente and M. Acharya. Intelligent jamming in wireless networks with applications to 802.11 b and other networks. In *MILCOM*, 2006.

[82] T. van Dam and K. Langendoen. An adaptive energy-efficient MAC protocol for wireless sensor networks. In *SenSys03*, 2003.

[83] K.-J. Wong and D. K. Arvind. SpeckMAC: Low-power decentralised MAC protocols for low data rate transmissions in specknets. In *REALMAN*, 2006.

[84] A. Wood, J. Stankovic, and G. Zhou. DEEJAM: Defeating energy-efficient jamming in IEEE 802.15. 4-based wireless networks. In *SECON*, 2007.

[85] H. Wu, K. Tan, Y. Zhang, and Q. Zhang. Proactive scan: Fast handoff with smart triggers for 802.11 wireless lan. In *Proceedings of IEEE INFOCOM*, 2007.

[86] Y. Xie, X. Luo, and R. Chang. Centralized psm: an ap-centric power saving mode for 802.11 infrastructure networks. In *Proceedings of IEEE Sarnoff Symposium*, 2009.

[87] W. Xu, W. Trappe, and Y. Zhang. Channel surfing: defending wireless sensor networks from interference. In *IPSN*, 2007.

[88] W. Xu, W. Trappe, Y. Zhang, and T. Wood. The feasibility of launching and detecting jamming attacks in wireless networks. In *MobiHoc*, 2005.

[89] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient MAC protocol for wireless sensor networks. In *INFOCOM*, 2002.

[90] W. Ye, F. Silva, and J. Heidemann. Ultra-low duty cycle MAC with scheduled channel polling. In *SenSys*, 2006.

[91] W. Ye, F. Silva, and J. S. Heidemann. Ultra-low duty cycle MAC with scheduled channel polling. In *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems, SenSys*, 2006.

[92] Z. Zeng, Y. Gao, and P. Kumar. Sofa: A sleep-optimal fair-attention scheduler for the power-saving mode of wlans. In *Proceedings of IEEE 31st International Conference on Distributed Computing Systems*, 2011.

[93] X. Zhang and K. G. Shin. E-mili: energy-minimizing idle listening in wireless networks. In *Proceeding of IEEE Transactions on Mobile Computing*, 2012.

[94] M. Zorzi and R. Rao. Geographic random forwarding (GeRaF) for ad hoc and sensor networks: energy and latency performance. *IEEE transactions on Mobile Computing*, 2003.

[95] M. Zorzi and R. Rao. Geographic random forwarding (GeRaF) for ad hoc and sensor networks: multihop performance. *IEEE transactions on Mobile Computing*, 2004.